16

ENE 08



REM

RETROEUSKAL '07 Y RETROMAÑÍA

hardware

INSTALACIÓN Y USO DE UNA UNIDAD DE DISCO DE 3" EN UN PC.

ensamblador

LENGUAJE ENSAMBLADOR DEL Z80 (III)

z88dk

SPLIB (SP1)

analisis

VIAJE AL CENTRO DE LA TIERRA
BETILED!
WIZARD OF WOR
STRANDED 2.5
JUSTIN

FIESTA 25 ANIVERSARIO
1 HORA
DESDE AQUÍ

opinión

EL MERCADO RETROINFORMÁTICO

http://magazinezx.speccy.org/

Diez meses desde nuestra última entrega, y ligeramente retrasados para celebrar el 25 aniversario del Spectrum, volvemos, justo al comienzo del nuevo año, con un buen cargamento de contenidos que esperamos que os gusten.

Pese a la tardanza, hemos querido dedicar la portada (compuesta por Juanje con el buen gusto y la calidad que le caracteriza) a los veinticinco años que cumplió nuestra máquina en 2007. Parece mentira cómo un producto de electrónica de consumo puede seguir despertando pasiones un cuarto de siglo después de su nacimiento, y habiendo transcurrido más de 10 años de su "muerte" comercial.

En la sección Panorama, hacemos balance de lo más relevante del año que recientemente nos ha dejado. Hardware, software, reuniones de usuarios, emulación, y todos esos ingredientes que hacen que los seguidores del mundillo den palmas con las orejas. No nos hemos olvidado de dos de las reuniones más importantes, RetroEuskal y Retromañía, a las que dedicamos un reportaje más completo en la sección REM.

La sección de análisis de juegos también es una retrospectiva del año pasado, ya que comentamos cinco de los juegos lanzados durante 2007 para nuestro entrañable Spectrum.

En la sección Hardware, explicamos cómo conectar una unidad de 3 pulgadas a un PC, paso previo imprescindible a la preservación del software editado en formato disco.

Los cursos de programación, si bien no con la periodicidad que todos desearíamos, siguen adelante. El curso de ensamblador disecciona todo lo relacionado con las instrucciones condicionales, mientras que en el de Z88DK lidiaremos con la SPLIB.

Como colofón, un artículo de opinión sobre la fauna que puebla los sitios de compraventa de material retro en Internet.

Esperamos que disfrutéis leyendo la revista tanto como nosotros haciéndola. No sabemos cuándo volveremos a encontrarnos, esperemos que no tengamos que esperar al próximo año, pero ¡nunca se sabe!

Redacción:

Miguel A. García Prada.
Javier Vispe.
Federico Álvarez.

Ilustración de Portada:

Juanje Gómez.

Colaboraciones:

Santiago Romero.
Pablo Suau.
José Leandro Novellón.
Juan Pablo López.
José Manuel Claros.



Maquetación PDF:

Javier Vispe Mur.

Contacto:

magazinezx@gmail.com

panorama

speccy.org: aniversarios varios

Este número de Magazine ZX dedica una especial atención al 25 aniversario del primer modelo de Spectrum lanzado al mercado. Sin embargo, no está de más recordar que el portal www.speccy.org también celebra onomástica. Si el mes de Abril fue testigo del anuncio realizado por Sinclair Research en el hotel Churchill de Londres, también lo fue del anuncio realizado por los miembros de Speccy.org en E.C.S.S. en 2002. Cinco años después el proyecto sigue en funcionamiento, recopilando, informando y dando hospedaje a cualquier iniciativa relacionada con el Spectrum y el resto de máquinas de la marca.

El nacimiento de speccy.org se vio reforzado en poco tiempo con las incorporaciones de webs como SPA2. Por entonces, Juan Pablo López Grao comenzó la tarea de recopilar software editado en España para Spectrum. A día de hoy, se ha convertido en la principal puerta de entrada de programas de nuestro país a la base de datos de The World of Spectrum.

En los dos últimos años la web se ha visto sometida a un profundo lavado de cara que está a punto de ver la luz. La espera merecerá la pena.



Otra de las webs veteranas que ha cumplido 5 años es SinclairQL.es. Hemos de dar la enhorabuena a Javier Guerra (Badaman) por seguir ofreciendo información y contenidos sobre el QL en castellano.

El aniversario también ha sido una excusa perfecta para que speccy.org siguiera ampliando servicios este año, y ha abierto un foro donde intercambiar opiniones. Desde aquí invitamos a la gente a registrarse y participar.

Cerramos este bloque de noticias relacionadas con el portal y las webs alojadas haciéndonos eco del lanzamiento de MHoogle 2.0. Tras 2 años en funcionamiento, Josetxu Malanda ha renovado esta herramienta de búsqueda para la revista Microhobby con grandes mejoras internas en su motor. Ahora es capaz de buscar, además de en los índices, en el contenido de cada una de las páginas de la revista.

Concurso "25 años de metal (y plástico)"

La pasada primavera se convocó desde speccy.org un concurso que se sumara a las iniciativas de los 25 años. El único requisito de participación era el envío de una fotografía relacionada con el Spectrum o la camiseta conmemorativa de los 25 años del mismo. El ganador fue Juan Francisco Jara, al que podéis ver en acción en la fotografía. ¡Enhorabuena, que disfrutes del premio!



Publicaciones

La revista ZX Spectrum Files llegó al número 11 durante el mes de Marzo, con un nuevo diseño. Además, su autor, Ignacio Prini editó un número homenaje a Microhobby durante el verano.

Repasando la actualidad inglesa nos encontramos con el retorno de ZXF. Colin Woodcock recuperó su revista con un número conmemorativo, concretamente el 11. El diseño homenajea a la revista Crash, e incluso está disponible en formato papel.

También hemos de hacer referencia a las entregas 2, 3 y 4 de RGCD, revista en formato electrónico preparada para descargar y grabar a un CD. La temática de esta publicación inglesa se centra en los juegos retro y los remakes. Algunos juegos de Spectrum como Cannon bubble, Blizzard's rift o Stranded 2.5 ya han pasado por sus artículos.

Radastan ha creado recientemente el fanzine Bytemaniacos, del cual ya han salido dos números. En el primero de ellos convoca otra edición más de sus concursos de programación en Basic.

Emulación

Estos últimos meses han sido bastante prolíficos en diferentes plataformas y sistemas operativos, apareciendo nuevos emuladores o versiones más actuales de viejos conocidos.

- En entornos Windows, Spectaculator llegó a la versión 6.30, siendo su novedad más destacable la compatibilidad con Vista.
- ZXSpin actualmente en la versión 0.666, ha centrado sus esfuerzos en el soporte de dispositivos IDE, mejoras en máquinas con TR-DOS y emulación del Plus D.
- Marat Fayzullin ha continuado con el desarrollo de Speccy para Windows, MS-DOS y Symbian. En la versión 1.5 ya se puede jugar por red.
- En cuanto a Eightyone, Mike Wynne ha vuelto a la carga. Con la versión test 0.52 la fiabilidad ha aumentado muchísimo, y ahora, por ejemplo, es posible trabajar muy cómodamente con las Compact flash formateadas para los plus 3e.
- Unreal speccy, especializado en la emulación de los modelos rusos ha llegado a la revisión 0.36.7, concentrándose el desarrollo en la eliminación de fallos y la mejora de general sound.
- DSP: en la parte que nos interesa, Leniad ha hecho hincapié en mejorar el soporte del Plus3, eliminación de errores, optimización de emulación y código, soporte de más periféricos y el nuevo formato PZX.

Camisetas 25 aniversario

Han sido necesarias tres tiradas de camisetas para satisfacer las peticiones realizadas. La iniciativa incluso ha tenido su hueco en medios de comunicación a nivel nacional. Desde estas líneas sólo podemos agradeceros una vez más vuestro interés. Este inesperado éxito ha repercutido en beneficio del portal speccy.org, permitiendo cubrir los gastos de alojamiento, y de RetroEuskal, ayudando a financiar las actividades que se realizaron en la edición de este año. Podéis conocerlas con más detalle en el reportaje que se incluye sobre el encuentro en este número.



Modelo posando con la camiseta

- En Specemu 2.5 han añadido emulación del Currah µSpeech y el "snow effect", soporte para las roms del interface 2 y velocidades para el Z80 de hasta 50 Mhz. También se han solucionado unos cuantos errores.
- Fuse (Linux y Mac) llegó a la versión 0.8.0 con añadidos en la emulación de hardware y con una versión beta para Windows.
- Los ordenadores Apple también han dispuesto de actualizaciones para ZXSP. De momento, la última versión pública es la 0.7.2.pre2. Para su funcionamiento necesita de la versión 10.4 o superior del sistema operativo y corre en Mac con procesador PowerPC o Intel.
- Los poseedores de una Nintendo DS en estos momentos tienen dos alternativas muy activas

y con una evolución muy rápida. SpeccyDS en la versión 0.3 ha añadido un nuevo interface, mejoras en la precisión de la pantalla táctil, controles redefinibles, implementación de instrucciones no documentadas del z80…

• La otra opción en la portátil de Nintendo es ZXDS. En pocos meses ha dado un salto de calidad enorme: soporte a los modelos de 128K (incluído Pentagon), carga de ficheros TAP y TZX, sincronización de la pantalla, menú y teclado virtual,… En estos momentos, la versión 0.7.0 es la opción más completa para esta consola.

• Metalbrain sigue trabajando en la GP2X. GP2Xpectrum 1.5b2 optimiza el rendimiento en la máquina con pequeñas correcciones a la temporización. Se ha reescrito el código de dibujar la pantalla, con lo que el borde está completamente emulado.

Plus 3e y ResiDOS

Garry Lancaster lanzó en Junio la versión 1.31. de las eproms del Plus3e solucionando un fallo presente en las roms originales que provocaba fallos en las cargas de programas. Mientras, su otro proyecto paralelo, ResiDOS, se actualizó a la versión 2.01. Aparte de la posibilidad de añadir nuevos módulos, se ha trabajado la funcionalidad con el interface ZXMMC+. En relación con los plus3e, Icabod está trabajando en una utilidad denominada 3eExplorer, que permite gestionar imágenes HDF de dispositivos de almacenamiento usados por dispositivos IDE en estas máquinas. De momento, ya se pueden explorar los HDF y extraer ficheros al PC, pero en futuras versiones también se podrá importar bloques o utilizar otros formatos como DSK. Drive image 1.32

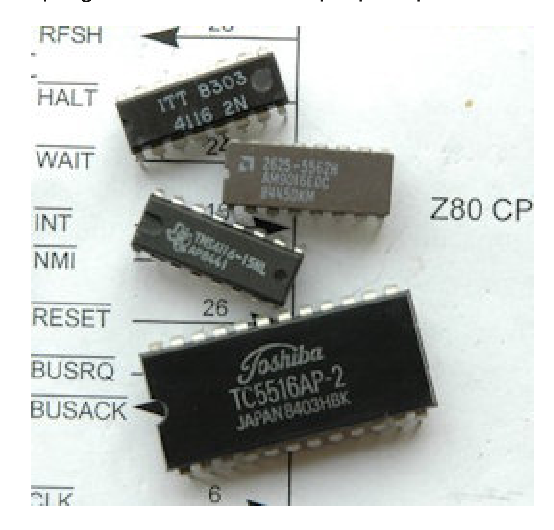
Pera Putnik, viejo conocido para los que trastean con el hardware del Spectrum, mantiene también una utilidad que permite extraer e importar imágenes HDF de dispositivos IDE, pendrives, tarjetas de memoria… Las últimas mejoras que ha incluido en Drive image consiguen velocidades de transferencia mucho mayores.

The your Sinclair Rock and Roll years: 1988

La serie de reportajes en vídeo de Nick Humphries ya ha llegado hasta el año 1988. Está disponible tanto para su descarga como para su visionado online.

Placa de diagnóstico de averias para Spectrum

Winston ha diseñado una placa que diagnostica averías comunes en los Spectrum. Este montaje permite averiguar de forma rápida qué componente del ordenador falla. Para ello, cuenta con una flash ROM con un programa que realiza las comprobaciones. La placa incluye 8 LEDs que muestran el progreso de los análisis. Su conexión es a través del puerto trasero de expansión del ordenador, y puede ser usada también como una ROM más. La memoria flash se puede programar a través del propio Spectrum.



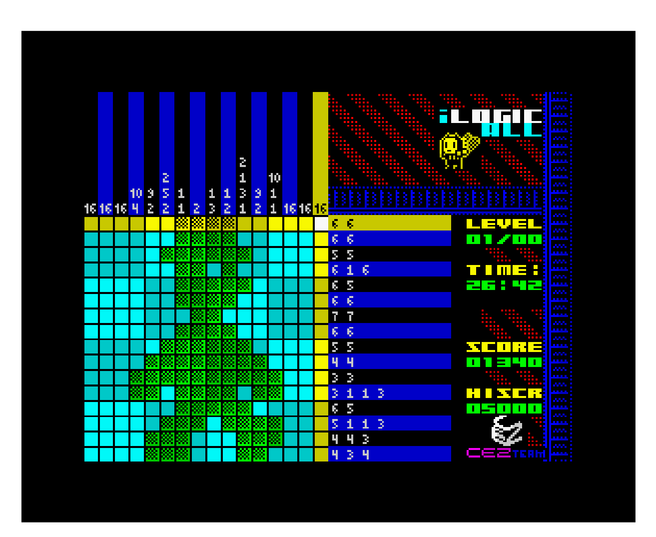
Nuevo software

Desde el último número de Magazine ZX se han sucedido unos cuantos lanzamientos y anuncios en el terreno del software.

• Topo S.XXI: Borrocop y Gandulf se han embarcado en crear un nuevo grupo de desarrollo. Su primer proyecto es la versión extendida de "Viaje al centro de la Tierra". En su momento, el juego fue comercializado sin dos fases que incluían las versiones de 16 bits y que, ahora, han recuperado en Spectrum.

• Computer Emu Zone Games Studio: han anunciado el desarrollo de más juegos, como UWOL, Quest for money, Ilogicall, Biniax o Subacuatic. Entre los proyectos finalizados y recientemente publicados

nos encontramos con el plataformas Phantomasa 2 y el puzzle Betiled!.

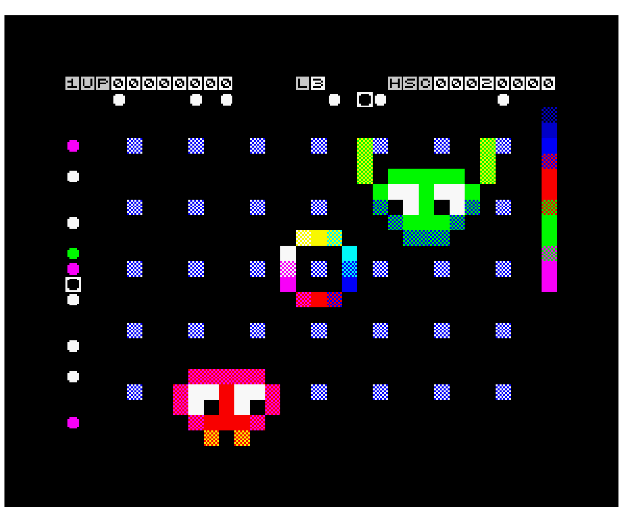


Ilogicall

• Cronosoft: la editora británica tampoco ha estado parada, y en los últimos meses ha sacado tres juegos más: Quantum gardening, Stranded 2,5 y Stronghold. Además Jonathan Cauldwell ha publicado recientemente la quinta entrega de Egghead y un minijuego con este protagonista para la web Eurogamer. Mientras, Bob Smith continúa con la programación de splATTR.

• Josep Coletas: los aficionados a las aventuras conversacionales pueden disfrutar de dos nuevos juegos de este autor: Witchcraft Edición Oro y Los extraordinarios casos del Dr. van Halen: Las calles del miedo.

• Octocom: recién salido del horno, Isotopía es el primer trabajo en Basic compilado de este grupo. Se trata de un homenaje a Atomino, puzzle de Psygnosis procedente de Amiga.



splATTR

Juegos inéditos

Tanto en el panorama nacional como en el internacional, se han recuperado diversos juegos que no llegaron a ver la luz en su época.

• Death pit: Durell lo anunció e incluso lo lanzó en CPC. Sin embargo, hasta hace poco tiempo no hemos podido verlo en acción, gracias a la colaboración de su programador, Clive Townsend.

• Time robbers: Micromanía llegó a analizarlo en uno de sus primeros números, pero sin embargo, nunca se supo nada más. Alberto Nadal facilitó una cinta al Trastero del Spectrum, de la cual se han recuperado dos versiones.

• New frontier: esta desarrolladora española es conocida por las conversiones que realizaron de juegos de 16 bits a máquinas más humildes con resultados excepcionales. En los foros de Computer

Conferencia sobre Retroinformática: Juegos, mentiras, y cintas de 8 bits

Eduardo Mena y Juan Pablo López, profesores de la Universidad de Zaragoza, dieron una conferencia a principios del mes de Mayo sobre los principales sistemas de 8 bits de principios de los 80. El evento se celebró en la Facultad de Informática de la Universidad Politécnica de Valencia y el Centro Politécnico Superior de la Universidad de Zaragoza.



EmuZone los miembros de la compañía han ido desvelando material inédito, como una versión del Sokoban original, y Shooting Range, un juego destinado a la pistola Gunstick que implementa un sistema de detección de disparos con una precisión de píxel.



Shooting range

Durell games está de vuelta

La mítica compañía británica hace unos meses que ha reabierto sus puertas con una continuación en PC de uno de sus primeros juegos: Harrier Atack II. Este lanzamiento hace uso de gráficos poligonales tratando de mantener el espíritu del juego original.

Ensambladores cruzados

Por si todavía queda algún programador despistado, desde el mes de mayo está disponible la actualización v1.07 RC5 "bugfix" del ensamblador cruzado de z80 SjASMPlus. Los cambios se han concentrado básicamente en la resolución de bugs.

También se puso a disposición pública la versión 1.7 de z88dk que, en conjunción con las rutinas para sprites de Alvin Albrecht, se han convertido en una herramienta muy valiosa para los que trabajan desarrollando para Spectrum. Para principios de 2008 se espera la siguiente entrega.

Demoscene

Los últimos meses han sido bastante activos en cuestión de parties en todo el continente europeo. Podemos nombrar la FOReVER eslovaca, la Breakpoint en Alemania, DiHalt y Chaos Constructions/ Antique en Rusia o Sundown en Gran Bretaña. A estas competiciones se suma la virtual Antique toy, y dentro de poco la Raww orgy que organiza icabod. Un buen sitio para poder consultar los resultados de todos estos eventos es: pouet.net.

CEZ retrocompo

El concurso de remakes organizado por Punisher ha llegado a su fin. Los resultados dieron como ganadores ex-aequo a Spirits de M.A. Soft y Camelot Warriors de BuHonero & Coelophysis.

Speccy Tour 2007

Alexandar Lukic se ha proclamado, por tercer año consecutivo, ganador del Speccy Tour. En esta ocasión la duración se redujo a dos meses, entre octubre y diciembre, exigiendo un menor número de juegos para clasificarse y admitiendo nuevos emuladores. Los títulos elegidos fueron: Dynamite Dan, Penetrator, All or Nothing, Deflektor, WS Basketball, Full Throttle 2, Cavelon, Tapper, Muncher y Deathchase.

Colabora con "The RZX archive"

Los archivos RZX son una especificación implementada originariamente en el emulador FUSE, y que permite visualizar partidas completas de juegos de Spectrum. La web "The RZX archive" se encarga de recopilarlos para ponerlos a disposición pública. Si tienes facilidad para acabar juegos de Spectrum, en especial españoles, quizás podrías colaborar con este proyecto para que haya una mayor representación del software hecho en nuestro país.

Javier Vispe

Links

3eexplorer: http://www.icabod.org/3eExplorer/
Bytemaniacos: http://www.bytemaniacos.com/
CAAD: http://www.caad.es/
Computer EmuZone: http://www.computeremuzone.com/
Cronosoft: http://www.cronosoft.co.uk/
Durell games: http://www.durellgames.com/
Eightyone: http://www.chuntey.com/eightyone/
Foro speccy.org: http://www.speccy.org/foro
FUSE: http://fuse-emulator.sourceforge.net/
FUSE windows: http://fuse-emulator.sourceforge.net/win32/fuse-win32-beta13.zip
Leniad DSP: http://leniad.cjb.net/dsp/index_es.htm
Mhoogle: http://mhoogle.speccy.org/
Octocom: http://www.octocom.es/
Placa de diagnóstico: http://www.alioth.net/Projects/Spectrum-Diag/
Plus 3e: http://www.worldofspectrum.org/zxplus3e/
Pouet: http://www.pouet.net/
ResiDOS: http://www.worldofspectrum.org/residos/
Retrocompo Computer Emu Zone: http://retrocompo.computeremuzone.com/
RGCD: http://www.rgcd.co.uk/
SinclairQL recursos en español: http://sinclairql.speccy.org/
Sjasmplus: http://sjasmplus.sourceforge.net/
SPA2: http://spa2.speccy.org/
Speccy: http://fms.komkon.org/Speccy/
Speccy.org: http://www.speccy.org/
Speccy tour: http://speccytour.blogspot.com/
Spectaculator: http://www.spectaculator.com/
The RZX archive: http://www.rzxarchive.co.uk/
Topo soft siglo XXI: http://www.topoxxi.creatuforo.com/
Unreal speccy: http://alonecoder.narod.ru/zx/
Your Sinclair Rock 'n' roll years: http://www.ysrnry.co.uk/
z88dk: http://www.z88dk.org/
ZXDS: http://zxds.raxoft.cz/
ZXF: http://zxf.magazine.googlepages.com/home
ZXSF: http://microhobby.speccy.cz/zxsf/pagina_1.htm
ZXspin: http://markboyd.myby.co.uk/zxspin.zip



En este número analizamos varios de los lanzamientos del pasado año: la Versión extendida del Viaje al Centro de la Tierra, Betiled!, la conversión del clásico de recreativas Wizard Of Wor, Stranded 2.5 y, por último, Justin.



VIAJE AL CENTRO DE LA TIERRA VERSIÓN EXTENDIDA

Topo Siglo XXI
2007

Carlos Arias, Rafael Gómez, Alfonso Fernández Borro, ACE, T.P.M., Antonio Moya, Manuel Ferreira Moreno, Sergio Vaquer Montes

Acción / Aventura / Puzzle
48K/128K
1 jugador

1989. El Spectrum está en plena efervescencia comercial. O al menos a nosotros nos lo parecía. Con las videoconsolas japonesas devorando las entrañas de la vieja Europa a través de su incipiente hegemonía, y los ordenadores de 16 bits convertidos en la aspiración de las generaciones más jóvenes, el mercado español todavía está anclado en los ordenadores europeos de 8 bits. Las desarrolladoras producen sus juegos para media docena de plataformas, pero las capacidades de los Amiga y Atari ST empiezan a pasar una factura que deja rastro en las versiones de los sistemas más modestos de 8 bits. Títulos que se arrastran en humildes Z80, limitaciones técnicas que se reflejan en el diseño, o incluso bajo el mismo nombre, juegos completamente diferentes en su desarrollo.

Sin embargo, la aparición ahora de la versión extendida de Viaje al Centro de la Tierra, nos dibuja un panorama completamente distinto. El recorte de las versiones de 8 bits, y no precisamente por falta de potencia, deja entrever cómo afectaba a la finalización de los juegos las presiones de las fechas de salida y la precariedad de la mal llamada para algunos industria española. La expansión realizada por Topo Siglo XXI añade al juego las fases que faltaban hasta completar las cinco inicialmente concebidas. Aprovecharemos pues para echar la vista atrás y analizar de forma global esta "remasterización".

Una portada extraordinaria, tiempo para leer las instrucciones de juego durante los cinco minutos que dura la carga más el interludio de definir teclas, una colorida introducción que desemboca en el hallazgo del mapa que indica nuestro viaje... ¿¿y un rompecabezas?? Siempre me pareció peculiar esta forma de empezar un título que prometía bastante acción. Si bien muchos juegos multicarga incluyen fases de transición con clara orientación al puzzle, éstas no suelen ser excesivamente complicadas y se incluyen entre otras con más peso específico. Aquí el mapa se convierte en un comienzo no muy atractivo a pesar de desplegar gráficos grandes que se desenvuelven perfectamente en pantalla. Tenemos la opción de cargar directamente la siguiente fase una vez que sepamos la clave de acceso al nivel dos, pero se trata de jugar todo, no de saltarnos partes, ¿no?

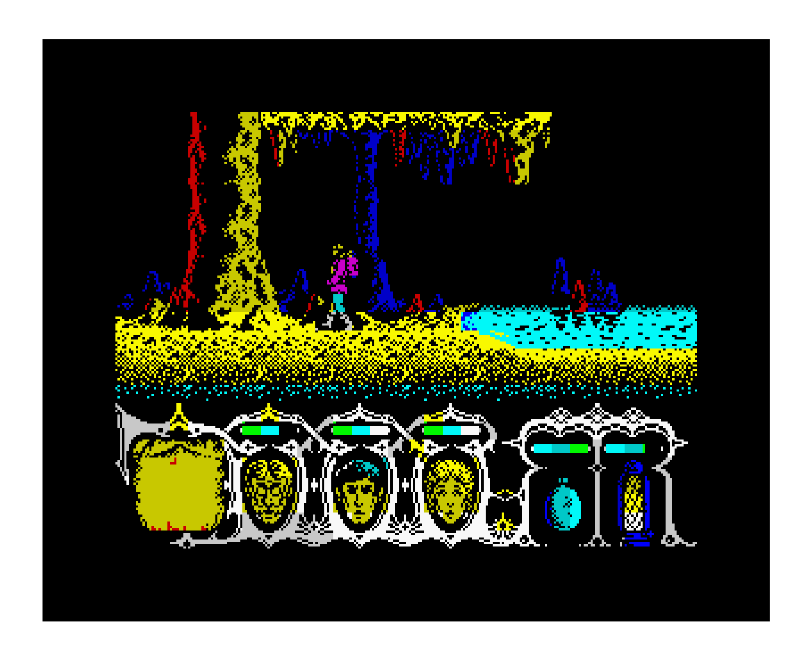
El acceso al interior del volcán nos propone un escenario mucho más atractivo. Un laberinto enorme que explorar para llegar a lo más profundo, repleto de enemigos y peligros en forma de precipicios, géiseres, arañas, murciélagos, lava... Debemos guiar a los tres personajes, cada uno con características propias, lo que indefectiblemente nos lleva a interactuar con ellos para llegar al final. Esta fase es la que más destaca a nivel gráfico, con sprites de un tamaño considerable y profuso uso del color. Los protagonistas se ven rodeados de un halo que evita la mezcla de atributos, un efecto que queda disimulado en parte por los fondos oscuros. Hay dos cuestiones que ensombrecen el resultado final de este nivel. Las caídas que

sufrimos cuando llegamos a lo más alto de las cuerdas (obligan a ser demasiado preciso con su utilización) y la situación de algunas trampas. Entrar en una nueva pantalla sobre un géiser y perder vida sin tener tiempo de reaccionar, penaliza al jugador en exceso sin tener posibilidad de respuesta.

La tercera fase discurre en la selva prehistórica del volcán. Los supervivientes de la partida habrán de enfrentarse a enemigos prehistóricos y evitar hundirse en las arenas movedizas. De nuevo Graüben, la chica, será la encargada de reponer la energía de sus acompañantes, que lucharán armados con lanzas contra los animales que les atacan. Tenemos tres rutas para elegir, en las cuales combinaremos esfuerzos para llegar al destino. El aspecto visual en esta ocasión reposa en el uso de dos colores, con escenarios y enemigos detallados y bien animados. Sin embargo, la jugabilidad se resiente. El uso de la lanza se torna poco efectivo, quedando en clara desventaja ante las acometidas de tiranosaurios y smilodones. La dosis de acción se ve ralentizada por la inclusión del componente estratégico de llevar a tres personajes, uno de ellos como fuente de energía.

La playa de las tortugas gigantes supone nuestra llegada a los niveles inéditos. Hemos de atravesar la población de reptiles evitando que nos aplasten en sus desplazamientos. En cierta forma, podríamos hablar de un Frogger multidireccional. Un planteamiento simple y clásico, que en parte se ve afectado por una rutina de detección de choques algo generosa a favor de la máquina. La parte técnica se sustenta sobre un scroll suave en todo momento, a pesar de mover simultáneamente unos sprites de tamaño considerable.

La fase final refleja la salida del









Originalidad	6
Gráficos	8
Sonido	7
Jugabilidad	6
Adicción	6
Dificultad	8

Links
http://descargas.topoxxi.com/

volcán. Es una erupción en la que debemos esquivar los salientes de la chimenea para llegar a la superficie sanos y salvos. Conforme ascendemos, la velocidad de desplazamiento será mucho mayor, con lo cual todo dependerá de los reflejos con los que movamos de izquierda a derecha la plataforma sobre la que nos encontramos. Con diferencia, es la parte más sencilla de terminar, siendo un colofón que nos puede dejar un poco fríos.

Viaje al centro de la Tierra se nos vendió como una superproducción del software patrio. Al margen de la veracidad del eslogan, con la perspectiva de la distancia en el tiempo, vemos que peca en aspectos de planteamiento de jugabilidad y de estructura. La mezcla heterogénea de géneros no acaba de funcionar debido a la dificultad desigual y la excesiva sencillez conceptual de determinadas fases. Alguna de ellas incluso nos puede costar más tiempo cargarlas que pasarlas.

Esto no es óbice para dejar de nombrar el buen trabajo realizado en el apartado técnico del juego. Gráficos coloridos, o, en las fases monocromas, muy trabajados, junto a desplazamientos notables de sprites en pantalla con scrolls muy suaves. La música original ejecutada a través del zumbador versionea una composición clásica siguiendo el estilo sonoro característico en Topo desde los tiempos de César Astudillo, mientras que en el menú de la versión extendida Sergio Vaquer opta por el chip de los 128K con una melodía apoyada en percusiones. El resto del juego se limita a contar con pocos efectos.

El paso del tiempo ha hecho más evidentes los defectos que en su día tenía el juego, y quizás haya desgastado sus virtudes. Sin embargo, hemos de admitir que esto ocurre con la inmensa mayoría del entretenimiento electrónico. Sólo unos pocos conservan su esencia inalterada; a pesar de verse superados tecnológicamente, siguen manteniendo su vigencia tanto desde un punto de vista jugable como histórico. En este caso, nos quedaremos con el segundo punto, y celebraremos el que se nos haya dado la oportunidad de ver cómo debería haber sido Viaje al centro de la Tierra desde un principio.

BETILED!

Computer EmuZone Games Studio
2007

Benway, Kendroock, Anjuel, WYZ, Augusto Ruiz

Puzzle
48K/128K
1 jugador

Bajo este nombre, Benway presenta su opera prima. Basado en el popular juego de flash Bejeweled! y su reencarnación como Zooo/Zoo-keeper en videoconsolas, nos encontramos con otro puzzle de combinar fichas que van cayendo desde lo alto de la pantalla. El objetivo es eliminar suficientes unidades de cada color antes de que termine el tiempo. Conforme avanzamos niveles, la dificultad aumenta, y contamos con una serie de items que nos ayudarán en nuestra labor. Sin embargo, deberemos tener cuidado con otros, ya que nos perjudicarán en nuestro cometido.

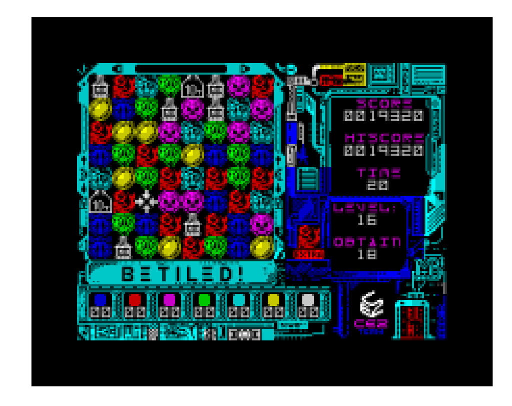
Contamos con tres modos de juego:

- arcade, donde se nos cuenta la historia que se desarrolla mientras jugamos.

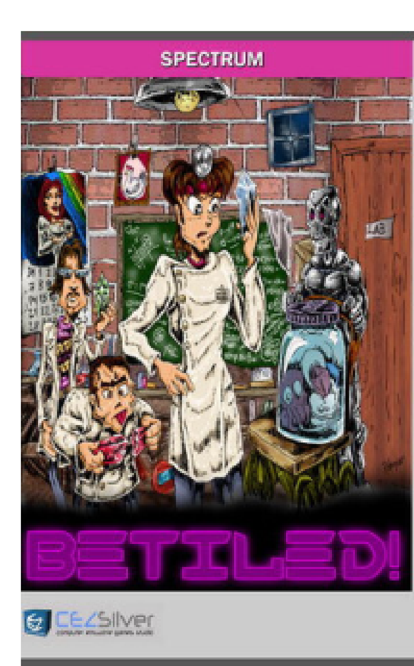
- normal, un "time attack!" donde se trata de hacer la máxima puntuación pero sin la aparición de items.

- especial, desbloqueable en el modo arcade, siendo idéntico al anterior pero incluyendo items.

El aspecto visual de Betiled! es algo irregular. Contrasta la excesiva sencillez de alguna de las imágenes estáticas que acompañan el juego con el mayor detalle de las de las transiciones del modo arcade. En este







aspecto, el abigarrado marcador transmite la sensación de querer evitar el vacío de la pantalla, lo que le da un aspecto caótico y posiblemente despiste al ojo a la hora de consultar las piezas que faltan por eliminar. En cuanto a los sets gráficos que desbloqueamos conforme pasamos determinados niveles, en general cumplen bien, aunque en varios de ellos hay algo de confusión entre las piezas y los items que nos dan ventajas o nos perjudican. El cursor también es algo complicado de localizar si le pierdes la pista en algún momento de la partida. Me temo que la exigua paleta de colores del Spectrum pasa factura...

En cuanto al movimiento y control del cursor, la adaptación ha sido perfecta y podemos manejarlo para hacer combinaciones rápidamente sin tener problemas de respuesta y fluidez. El pilar fundamental del juego cumple con nota.

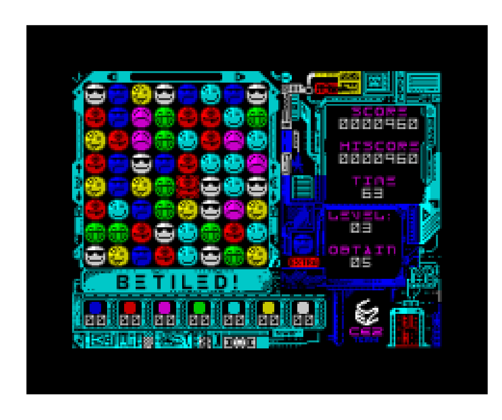
El sonido de Betiled! hace uso exclusivamente del chip AY-3-8912. La música acompaña al menú y los interludios, de forma que nos presenta a los diferentes personajes que hilan la historia del juego reflejando su personalidad. Mientras jugamos, sólo disponemos de efectos de sonido bastante futuristas. La elección no es mala, algo que seguro opinan los que dispongan de Nintendo DS y una copia de Zookeeper. Seguro que me entienden.

La dificultad es progresiva, aunque se nota un salto en la misma, tornándose bastante más exigente a partir del ecuador del juego. En esos momentos, sólo los que vayan haciendo combinaciones sin pausa podrán tener posibilidad de seguir adelante. Esto, y el planteamiento tan simple del concepto de juego, hacen que Betiled! comparta la adicción que provocan sus hermanos. Es uno de esos juegos que llamamos tontos. Una idea simple que no hay forma humana de dejar de jugar una vez que se ha empezado, a no ser que lo tuyo no sean los puzzles.

Originalidad	5
Gráficos	6
Sonido	7
Jugabilidad	7
Adicción	7
Dificultad	7

Links

http://cezgs.computeremuzone.com/ficha.php?id=19







WIZARD OF WOR

Weird Science Software
2007

Pgyuri, Naray, Edy

Laberinto
48K
1 ó 2 jugadores

Originalidad	4
Gráficos	5
Sonido	5
Jugabilidad	6
Adicción	6
Dificultad	8

Links
http://wss.sinclair.hu/wizard_of_wor.php

Hacía tiempo que no sabíamos nada de este grupo de desarrollo húngaro. Su regreso a la actualidad viene originado por la traslación de un arcade antediluviano de Bally/Midway, concretamente, de 1980. Si bien fue convertido a otros sistemas domésticos como Commodore 64 o Atari 2600, es ahora cuando podemos cargarlo en nuestros Spectrum. Veamos cuan fiel ha sido el resultado.

Nosotros tomamos el papel de los Worriors, encargados de eliminar todos los monstruos que infestan los laberintos que componen el desarrollo del juego: burwors, garwors, thorwors, worluks y el que da nombre a la aventura, el "Wizard of Wor".

Los comienzos son sencillos, y a pesar de encontrar una buena cantidad de burwors no tendremos problemas en eliminarlos. La cosa se complicará conforme empiecen a llegar enemigos de mayor entidad, que cuentan con la posibilidad de hacerse invisibles. Podemos usar un portal entre los extremos del laberinto que se va reactivando conforme se usa, aunque también pueden pasar los monstruos. La eliminación de los worluks nos dará acceso a la denominada "Double score dungeon", que como su nombre indica, permite conseguir el doble de puntos en la siguiente mazmorra.

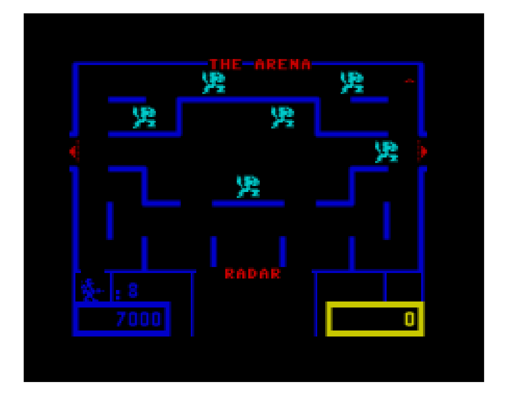
Wizard of Wor da la posibilidad de jugar dos personas simultáneamente. En el arcade original cuando sólo participa un jugador humano, el otro es controlado por la máquina. En la versión de Spectrum el modo de un sólo jugador se limita a mostrar un sólo worrior en pantalla.

Nos encontramos con un juego que ha tratado de ser lo más fiel posible al original. Gráficamente es muy similar a lo que se pudo ver en 1980, contando con que, a pesar de lo primitivo del hardware de la placa recreativa, determinados efectos visuales escapan de las capacidades del Spectrum. El movimiento resulta algo más lento, lo que resta capacidad de reacción a la hora de atacar o escapar de nuestros enemigos.

El apartado sonoro es bastante espartano, limitándose a melodías simples entre laberinto y laberinto y unos pocos efectos de sonido durante la partida a través del zumbador. Pero es que el mayor lujo que se permite el Wizard of Wor de Bally/Midway, es ir acompañado de un buen puñado de voces sampleadas.

La dificultad del juego se ve acentuada en parte por el control lento de nuestro personaje. En ocasiones no tendremos tiempo de reaccionar si la posición del mismo es en mitad de un cambio de dirección. A pesar de este inconveniente, la conversión mantiene la esencia del arcade.

Esta fidelidad tan escrupulosa implica una parte positiva y una parte negativa. Tenemos la posibilidad de disfrutar de un juego de corte clásico.





Nada de florituras, sólo una mínima simbolización en pantalla aderezada con acción pura y una pizca de estrategia. Sin embargo, llega con más de veinte años de retraso, con las comparaciones odiosas que ello conlleva. Sin añadidos o reinventado de alguna forma, pierde la frescura que hubiera tenido de salir hacia 1983-84. Parece como si alguien nos lo hubiera enviado por medio del De Lorean. Sólo que en 1987 también nos habría dejado con sensación de déjà vu.

En definitiva, el que busque jugabilidad primigenia y no quiere recurrir a M.A.M.E., puede quedar satisfecho con lo nuevo de Weird Science Software. Encontrará un acabado correcto, guiños a reencarnaciones previas y laberintos oscuros donde acechan misteriosas y monocolores criaturas. Al fin y al cabo, estos húngaros no siguen las modas. Crean la suya propia.

Dos años después de publicar Stranded, Bob Smith nos hace llegar su secuela. El argumento nos habla de Tyche, un pequeño ser que ha sido aprisionado por un tal Moosh. No contento con ello, también está apoderándose de los recursos de su mundo y esclavizando a sus pobladores. Tras escapar de su cárcel, Tyche se dispone a retornar para liderar la revuelta contra el tirano.

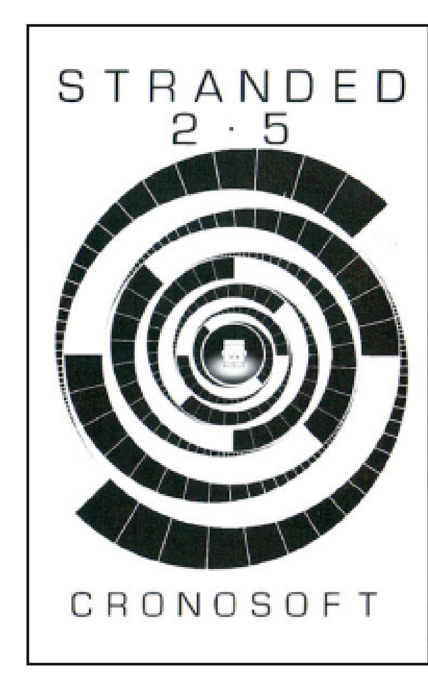
Este regreso se hace a través de seis mundos divididos en ocho niveles. Nuestro camino se compone de bloques que desaparecen conforme pasamos sobre ellos. Para completar la fase tendremos que eliminarlos todos y llegar a la salida. Al margen de las plataformas normales, encontraremos otras que precisarán una mayor insistencia para desintegrarse, bloques fijos, intermitentes o de dirección predeterminada e interruptores que habilitarán suelos ocultos.

La saga Stranded se podría decir que es un híbrido entre Make trax y Check Man. Debemos pasar por todos los bloques que componen el laberinto mientras desaparecen, con la peculiaridad de que el único enemigo somos nosotros contra el tiempo. Si no pensamos bien nuestros movimientos, quedaremos irremisiblemente encerrados sin posibilidad de llegar a la salida.

Lo primero que nos salta a la vista tras cargar el juego son los menús y marcadores, claramente inspirados en los trabajos de Costa Panayi. El grueso de los gráficos son los bloques por los que discurre la acción. Su diseño es sencillo, jugando con el brillo para dar cierta sensación de tridimensionalidad. Esto, unido al acertado uso del color, da un resultado final muy efectivo. Como puntos negativos, la mezcla de atributos que se produce al paso de Tyche y la simpleza del fondo, al que quizás se le podría haber dado más variedad.

Stranded 2.5 hace un uso extensivo del chip de sonido de los modelos de 128K. Cada mundo tiene asignada una melodía, y cada transición su correspondiente "jingle". El trabajo de Lee Ducaine destaca de forma brillante. La carga del juego en un 48K nos dejará sólo con los efectos de sonido, correctos sin más.

El primer mundo, Lowdrone, es una especie de tutorial que nos sirve para hacernos con el sistema de juego. La dificultad es baja, cosa que empieza a cambiar a partir del segundo mundo.



STRANDED 2.5

Cronosoft
2007

Bob Smith, Lee Du-caine

Puzzle
48K/128K
1 jugador

Originalidad	6
Gráficos	7
Sonido	9
Jugabilidad	7
Adicción	7
Dificultad	7

Links

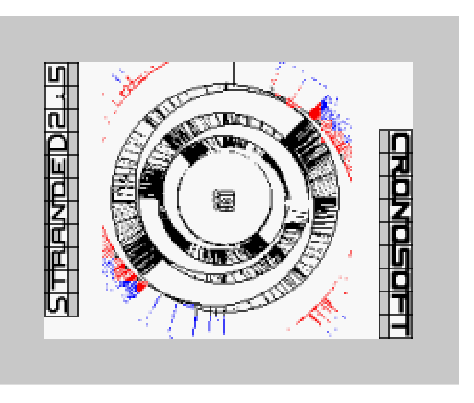
http://www.worldofspectrum.org/infoseekid.cgi?id=0017731

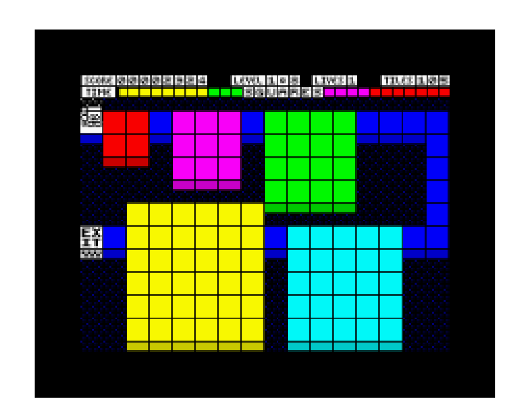
A la venta en:
http://www.cronosoft.co.uk/

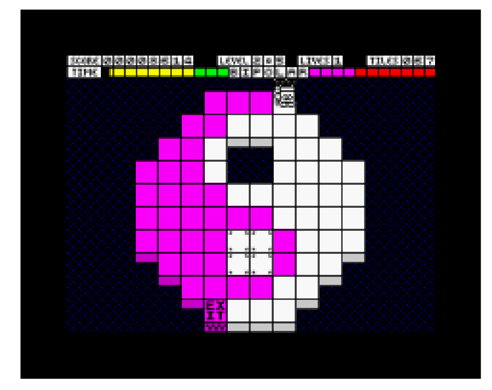
Descubriendo las reglas de planificación básicas, tendremos mayor facilidad para avanzar, aunque en algunas fases nos tocará poner la pausa y pensar detenidamente el recorrido. Disponemos de intentos ilimitados para salir dentro del tiempo establecido, aunque llega un momento en que los segundos que nos restan son insuficientes para poder terminar.

Juzgar la adicción de este título es complicado, puesto que dependerá de la implicación que tenga el jugador. Si lo suyo es el gatillo fácil sin parar a pensar el próximo movimiento, no tardará mucho en pulsar el botón de reset. Si le gusta pararse a pensar como resolver el rompecabezas, tiene garantizada una tarde entretenida. O el tiempo que necesite, porque el sistema de claves permite continuar en otro momento.

Para cerrar este análisis podemos dejar caer la idea de que Stranded 2.5 no es un plato de cocina moderna. Más bien es una receta de la abuela: tradicional, efectiva y que deja un buen sabor de boca.







JUSTIN

CNG soft
2007

César Nicolás González, Miguel Ángel Montejo

Aventura
48K/128K
1 jugador

Los juegos isométricos no abundan en el mercado actual del videojuego, pero tampoco son habituales de ver entre los lanzamientos de plataformas clásicas. El diseño de los mismos para hacerlos atractivos y divertidos, y la complejidad técnica de su programación no facilita la cosa. Incluso cuando se elimina esta barrera, como con el mítico 3D Game Maker, no cunden los buenos resultados. Hace dos años que este homenaje al Dustin de Dinamic salió para los Amstrad CPC. Los propietarios de un Spectrum han tenido que esperar hasta 2007 para tener el juego disponible.

Justin se propone asaltar la Gran Mansión como otro reto más de su dilatada carrera: tiene una hora de tiempo para hacerse con el oro de las cuatro cajas fuertes a base de dinamita. El acceso a algunas zonas de la casa están bloqueadas por puertas. Tendremos que buscar por el mapeado las llaves que nos permitan abrirlas y continuar la búsqueda. La tarea no va a ser sencilla, puesto que nos vamos a encontrar con guardias de seguridad y peligros en forma de arañas, ratones, plantas, suelos falsos...

¿Qué nos hemos encontrado en nuestro paseo por la mansión? Sin caer en debates pueriles sobre si "mi

CPC es mejor que tu Spectrum", la versión original tiene un acabado más cuidado. La menor resolución en Amstrad no impide que a través del color se consiga una aspecto envidiable. En el Spectrum, debido a sus limitaciones, se ha optado por el uso de tramas. Sin embargo, el resultado final produce confusión en algunos decorados donde no se distingue bien entre paredes, plataformas y objetos.

Una cuestión que sorprende es la elección de cambiar la perspectiva en que se muestra tanto a Justin como a los guardias, que no se corresponde con la isométrica utilizada en el resto del juego. La inspiración en sprites de Dustin (y algún otro juego) confunde en el manejo del personaje, hasta que uno logra acostumbrarse a esta peculiaridad.

En el apartado sonoro, nos encontramos con la misma música de los CPC, algo de esperar cuando se comparte chip encargado de esta tarea. Se opta por usar melodías ya conocidas popularmente, y que nos acompañan durante los interludios y el menú. En particular, esta última tiene un ritmo demasiado frenético. En la partida, tenemos efectos de sonido como pasos, explosiones o el salto, al más puro estilo de los clásicos de Jon Ritman. Eso sí, la carga en un 48K deja al juego completamente mudo.

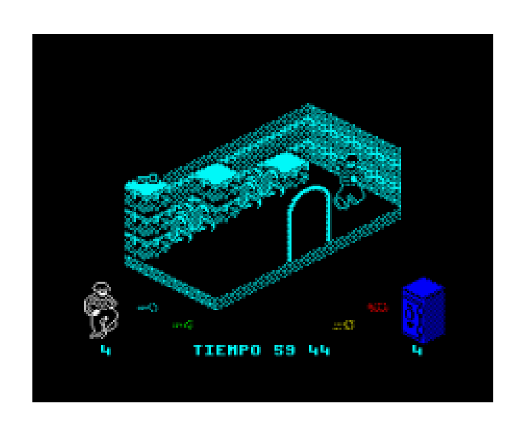
Tenemos 99 habitaciones que visitar en la mansión y cuatro vidas para terminar la aventura. Cuando abramos cada caja fuerte obtendremos vidas extra, algo de agradecer en juegos de este tipo, donde la dificultad es bastante alta. En parte, se podría haber calibrado algo más con la cuestión de la perspectiva del ladrón. También se echa en falta una referencia en los suelos que ayude a situarnos con respecto a los enemigos o trampas, así como las diferentes alturas que hay en algunas salas.

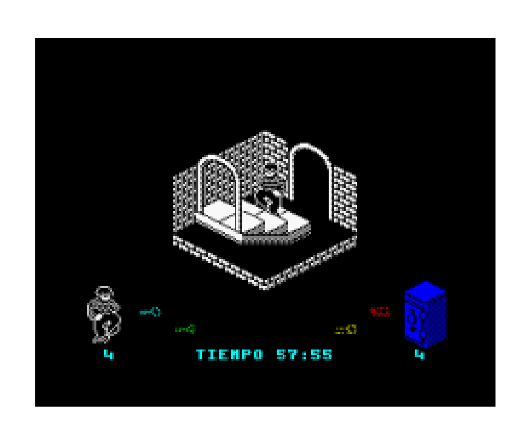
Justin agradará a los seguidores de las aventuras isométricas, y es un proyecto de mérito por la dificultad que lleva crear un juego de estas características. Sin embargo, en algunos apartados se echa en falta el haber dedicado más tiempo a su puesta a punto. Esos detalles hubieran significado el poder haberse acercado a los grandes del género. Esperemos que si hay una continuación, podamos concluir su análisis con una frase tan contundente.



Originalidad	7
Gráficos	6
Sonido	6
Jugabilidad	6
Adicción	7
Dificultad	8

Links
http://bytemaniacos.com/html/zxspectrum/justin.html





Javier Vispe

hardware

Instalación y uso de una unidad de disco de 3" en un PC

Si hay una gran contienda pendiente en cuanto a la preservación de software de Spectrum se refiere, podríamos resumirla bajo cuatro siglas: EDSK. El formato definido por Doewich, Thacker y Vieth cubre, aparentemente, todas las expectativas de los entusiastas del +3, al soportar toda protección de disco conocida para las tres pulgadas. Sin embargo, no son muchos los que tienen una disquetera de 3" instalada en su PC, requisito imprescindible para crear imágenes perfectas de discos de 3" en formato EDSK, así como para volcar éstas de nuevo a disco. Con este artículo trataremos de animar al lector a dar el paso que le permitirá preservar perpetuamente intactos sus discos de 3".



Antes de ponernos manos a la obra, conviene advertir que ni el autor de este artículo, ni SPECCY.ORG, ni la redacción de Magazine ZX, se hacen responsables de los posibles daños que pudiese ocasionar en su PC u otros componentes al seguir esta guía. Particularmente el lector debe recordar que bajo ningún concepto puede conectar la disquetera de 3" a su PC simultáneamente a otras unidades de disco flexible (e.g., la de 3 1/2"), so riesgo de provocar daños irreversibles en la controladora o unidades de disco.

Propósito

Con la instalación de una disquetera de 3" en nuestro PC perseguimos, principalmente, dos cometidos:

1. Poder preservar discos de 3", y especialmente los protegidos, en un archivo .DSK (imagen de disco en formato EDSK).

2. Poder grabar discos de 3" a partir de imágenes como las anteriores, de tal modo que seamos capaces de utilizarlos directamente en nuestro +3.

Con respecto al segundo punto, existe la errónea creencia de que conectando una disquetera de 3 1/2" al +3 (véase artículo relacionado de MagazineZX referenciado más adelante, en el apartado de enlaces), y copiando de 3 1/2" a 3" por medio de un copión, podemos igualmente lograr tal propósito, pero

lo cierto es que con ciertas protecciones de disco esto no funciona. Por el momento, la única manera de conseguir volcados perfectos de un DSK es conectando una unidad de disco de 3" a nuestro PC. En ello vamos a centrar nuestro esfuerzo de aquí en adelante.

Material requerido

• Un PC. Si simplemente pensamos utilizar CPDRead/CPDWrite, nos basta con un PC viejo, al que le instalaremos MS-DOS. Un 486 o un Pentium lento es ideal. En caso de que queramos utilizar CPCDiskXP, necesitaremos un PC que pueda soportar un Windows 2000 o XP. Si estamos dispuestos a prescindir de la disquetera de 3 1/2", podemos adaptar el PC de sobremesa que utilicemos habitualmente. Hay que tener presente que la unidad de 3" y la de 3 1/2" nunca podrán estar simultáneamente conectadas.

• Una faja de disquetera de PC convencional, con al menos un conector hembra para disquetera de 3 1/2" (34 PIN IDC). La misma faja que lleve el PC bastará, con toda seguridad.

• Tijeras y/o cutter. Será necesario realizar algunos cortes en el cable plano.

• Un conector IDC de 26 pines. En caso de apuro, podemos sacarla de una faja de diskettera de un +3.

• Una disquetera de 3". Igualmente, podemos extraerla de un +3 o CPC6128.

Afortunadamente, no es necesario que destripemos un +3 si no queremos, sino que podemos acudir a alguno de los múltiples vendedores que existen en la red, como el afamado John R. P. King, especialista en las 3 pulgadas:

http://www.pcwking1.netfirms.com/plus3.html

(no, no es colega, ni cobramos comisión por la publicidad ;-))

Otra opción es, por supuesto, comprar una unidad externa de disco Amstrad FD-1 y conectarla al PC. De ese modo te evitarás realizar el montaje del que versa este artículo, ya que no necesita ningún tipo de adaptador (el conector es IDC de 34 pines).



foto: José Leandro Novellón

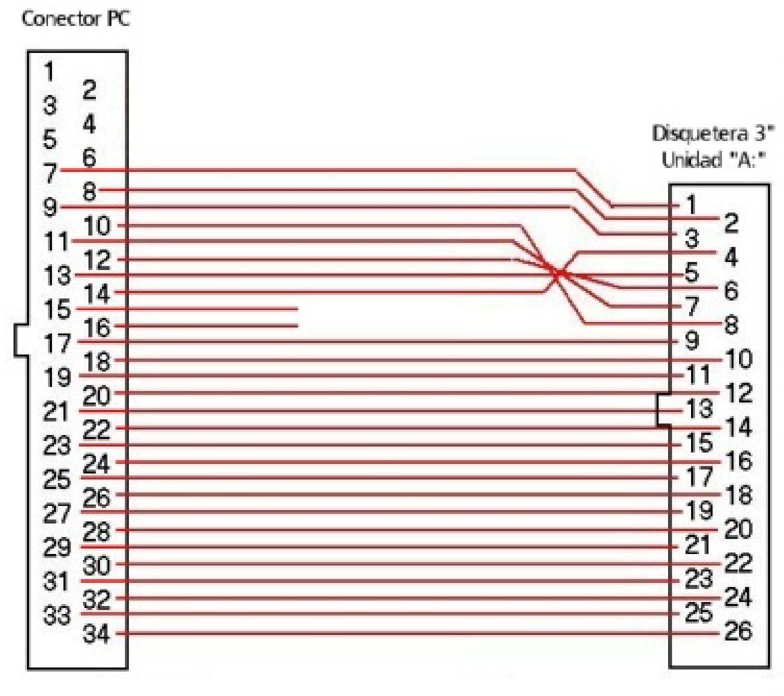
Realización del cable

Esquema del cable

La disquetera del PC tiene más señales que la disquetera de 3". Será necesario adaptar el cable para poder conectar la disquetera de 3". Por suerte, las modificaciones son pequeñas y la mayoría de señales y pines coinciden en ambos casos.

El cable que fabricaremos aquí permitirá conectar una disquetera de 3" como unidad "A:" en un PC.

El esquema del cable es el siguiente:



Pin 3.5"	Pin 3"	Description
2	NC	Density Select
4	NC	Reserved
6	NC	Reserved
8	2	Index
10	8	Motor Enable A
12	6	Drive Sel B
14	4	Drive Sel A
16	NC	Motor Enable B
18	10	Direction
20	12	Step
22	14	Write Data
24	16	Floppy Write Enable
26	18	Track 0
28	20	Write Protect
30	22	Read Data
32	24	Head Select
34	26	Disk Change

Nota:
Todos los pines impares son GND

Construcción

Con ayuda de las tijeras, cortamos el conector IDC 34 que está después del cruce de cables.

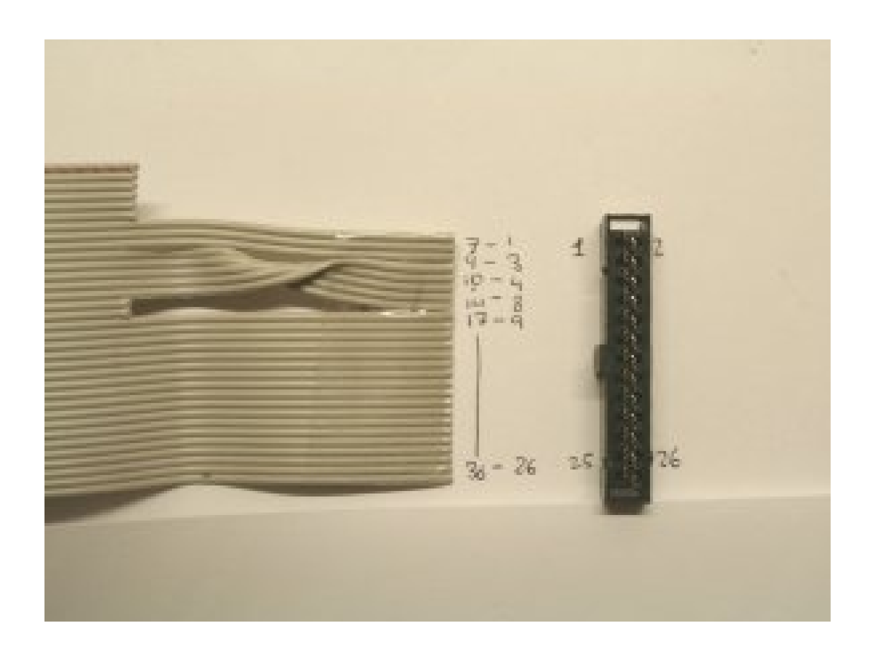
Si nos fijamos en el cable, veremos que hay un cable pintado de rojo. Esto nos indica cual es el hilo número 1.

Pasamos a eliminar los cables que no necesitamos. En este caso, los cables que van del 1-6 y la pareja 15-16.

Para juntar los cables, antes de presentarlo sobre el conector IDC de 26 pines, podemos usar un poco de celo o cinta aislante. Al cortar algunos cables, el resto se resisten a quedar perfectamente alineados.



fotos: José Leandro Novellón



Presentamos el cable sobre el conector IDC 26 el cual solo puede ser puesto en una posición. En el conector hay una flecha en un extremo que indica cual es el cable número 1. Si no encontramos la flecha nos podemos valer de la muesca grande en el centro del conector. En la foto se puede ver que el cable número 1 es el primer cable empezando por arriba y que no se ve la muesca (está en la otra cara)

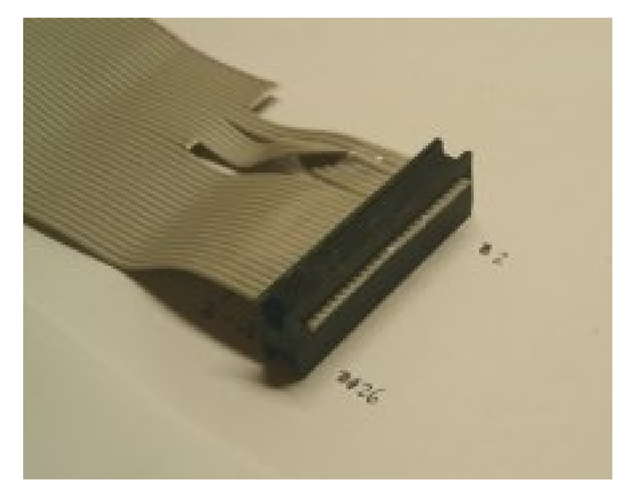


foto: José Leandro Novellón

Comprobaremos que los cables quedan encima de los dientes del conector IDC 26. La pieza que cierra ayuda pues tiene una ligeras marcas.

Con la ayuda de un martillo, iremos golpeando la pieza que sujetará el cable contra el conector. Poco a poco veremos como va bajando y el conector quedará cerrado del todo.



foto: José Leandro Novellón

El cable ya está finalizado. Si se dispone de un polímetro, es aconsejable comprobar que hay continuidad entre ambos el conector de PC y el de la disquetera de 3".

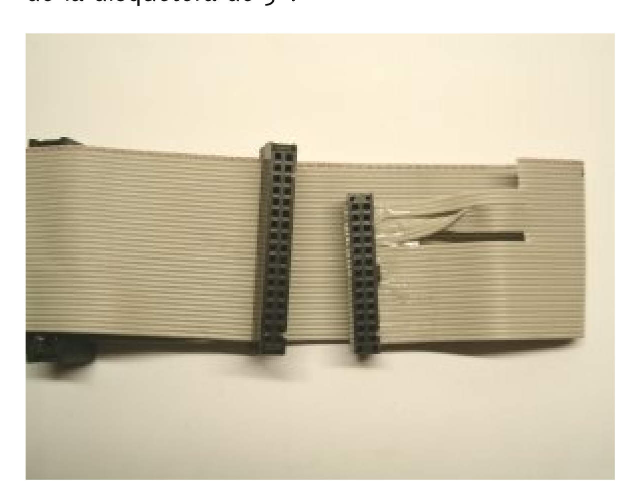


foto: José Leandro Novellón

Conexión de la diskettera al PC

En primer lugar, insistimos en que la unidad de 3 1/2" debe estar desconectada del PC. Es importante recordarlo: ambas unidades nunca pueden coexistir.

Listo el cable de datos, es necesario adaptar el conector de alimentación del PC que irá a la toma de la disquetera de 3". En la siguiente imagen, se muestra el conector de alimentación de una disquetera de 3 1/2" (izquierda) y el de una de 3" (derecha).

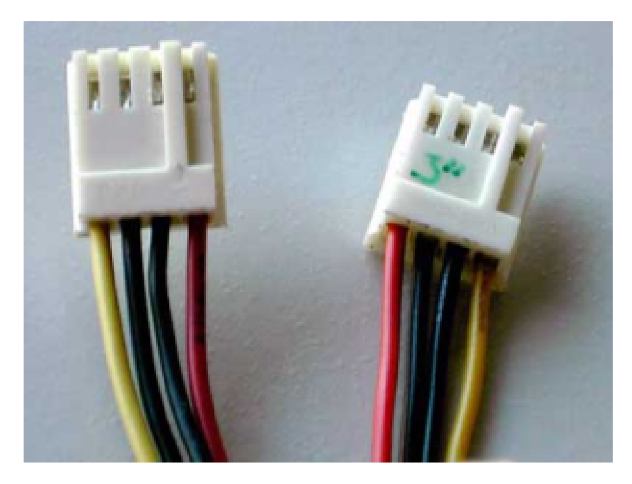
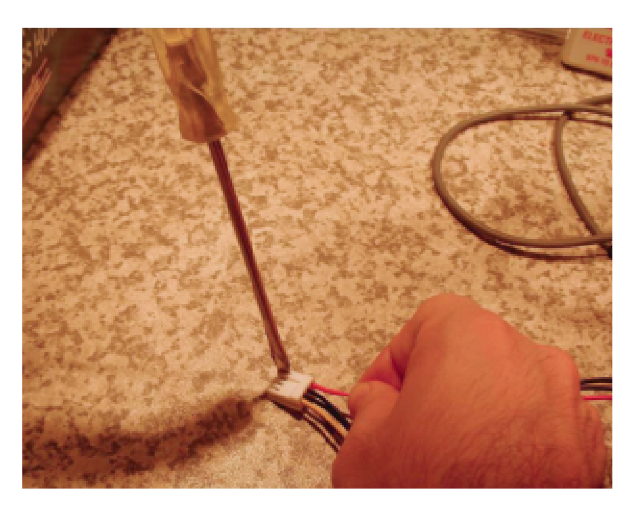


foto: Jaime González Soriano

Como puede verse, la toma de alimentación de 3" lleva intercambiadas las señales de +5v (cable rojo) y +12v (cable amarillo). Por tanto, deberemos intercambiar dichos cables (rojo y

amarillo) en el conector de nuestro PC. Las señales de tierra (cables negros) pueden quedar igual.

En la propia imagen puede observarse como el conector cuenta con cuatro pequeñas ranuras, a través de las cuales se ve la terminación metálica de cada uno de los cables. Para poder extraer uno de ellos, basta con introducir un destornillador plano por la ranura correspondiente, de manera que desplace levemente la terminación metálica del cable hacia abajo, y luego tirar del cable hacia fuera. No es necesario apretar fuerte con el destornillador; se trata simplemente de aplicar la presión de forma correcta.



Cómo intercambiar los cables rojo y amarillo con ayuda de un destornillador

Finalmente, conectaremos la disquetera a la controladora de disco del PC (FDC) empleando el cable de datos que hemos desarrollado, y enchufaremos el recién adaptado conector de alimentación a la correspondiente toma de la disquetera. A continuación, arrancamos el PC, entramos en la configuración de la BIOS y, si ésta nos lo permite, definimos la unidad de disco como 5 1/4" 360 Kb. También podemos quitar la unidad A: de la secuencia de búsqueda de dispositivos de arranque (bootup sequence) ya que carece de sentido arrancar el PC con un disco de 3".

Nótese que si la luz de la disquetera permanece continuamente encendida será muy probablemente porque hemos conectado el cable de datos al revés.

Configuración y puesta a punto del PC

Para manipular discos de 3" existen tres herramientas: CPD TOOLS (MS-DOS), CPCDiskXP (Windows) y SamDisk (Windows, aún en versión beta), cada una de ellas con sus pros y contras, que serán analizados más adelante. En caso de querer utilizar únicamente una de estas herramientas, bastará con tener instalado el sistema operativo correspondiente (MS-DOS o Windows). Pero si queremos poder utilizar las tres, algo que a día de hoy (diciembre de 2007) es más que recomendable, será imprescindible configurar un arranque dual (dual-boot). Esto implica el consiguiente particionado de nuestro disco duro (a menos de que estemos dispuestos a sacrificar un disco duro, destinándolo íntegramente a MS-DOS).

Si ya contamos con un sistema Windows, ¿realmente es necesario hacer una partición con MS-DOS? Bien. La cuestión es que, hasta donde conocemos, no es posible ejecutar satisfactoriamente CPD TOOLS desde el símbolo de sistema en Windows NT. Desgraciadamente, el S.O. interfiere con las operaciones a bajo nivel que las CPD TOOLS realizan con la disquetera. Y, dado que no tenemos disquetera de 3 1/2" en el PC, puesto que no pueden estar ambas conectadas al mismo tiempo, tampoco podemos emplear un disco de arranque con MS-DOS. Así pues, asumiremos que vamos a tener una pequeña partición con MS-DOS para las CPD TOOLS (puede ser FAT16 ó FAT32), y otra con Windows 2000/XP/..., desde la cual ejecutaremos CPCDiskXP o SamDisk (y que será FAT32 o NTFS).

Una de las cosas que primero debemos tener en cuenta es que, para que MS-DOS pueda arrancar, es necesario que se encuentre alojado en el primer disco duro, en una partición primaria. Suponiendo que no queremos tener discos duros dedicados, sino que vamos a particionar el disco principal, esto implica que tendremos al menos dos particiones primarias: la de Windows y la de MS-DOS.

Si vamos a partir de cero, podemos particionar el disco por medio de FDISK, instalar después MS-DOS, y a continuación instalar Windows 2000/XP, y de ese modo tendremos arranque

dual. La partición que se encuentre físicamente en primer lugar será la que reciba la letra de unidad C: en Windows, aunque el S.O. se encuentre instalada en la otra (¡ojo con esto!).

En cualquier caso, a menudo la situación es diferente: tenemos Windows 2000/XP instalado, y lo que queremos es habilitar una nueva partición para instalar MS-DOS sin destruir lo que ya tenemos. Una opción es utilizar una herramienta como Partition Magic, si disponemos de la correspondiente licencia. La otra, utilizar software libre, para lo cual es útil seguir la guía de TR-LOG incluida en la sección de enlaces, y que se resume en tres pasos: (1) ejecutar un CD de arranque de Linux que incluya un gestor de particiones como GParted o QTParted, y con él reparticionar el disco, (2) instalar MS-DOS 7.1, también desde CD auto-arrancable, y (3) instalar un gestor de arranque (boot loader) como puede ser GAG. Se trata de un proceso bastante mecánico y sencillo.

El interés de instalar el denominado MS-DOS 7.1 (una versión no oficial, por cierto) radica no sólo en el soporte en que es distribuido (CD autorrancable) sino en la posibilidad de dar soporte a FAT32 y nombres largos, lo que será muy útil si pretendemos manipular la partición de MS-DOS desde Windows (y realmente así será, con el fin de transferir los DSKs de una a la otra - recordemos que no tenemos disquetera para poder moverlos; y además, si la de Windows es NTFS, dicha partición será invisible desde MS-DOS). En ese sentido, conviene apuntar que, si la partición de Windows tenía originalmente asignada la letra de unidad C: en el sistema, deberíamos crear la partición de MS-DOS físicamente a continuación (y no antes) de ella, pues de otro modo será invisible desde Windows. No obstante, tampoco debería estar más allá del cilindro 1024, de acuerdo con algunas fuentes.

Manipulación de discos de 3"

Para volcar discos a DSK, así como para pasarlos de DSK a disco, existen tres herramientas en funcionamiento: la más antigua pero todavía "la reina", las CPD TOOLS, funcionan únicamente sobre MS-DOS. Para suplir la carencia de una herramienta sobre Windows nace CPCDiskXP, disponible para Windows 2000 / XP y en adelante. Y en este artículo os presentamos también una herramienta que se encuentra aún en fase beta, pero que vendrá a ampliar el parque de herramientas para Windows 2000 / XP, y posiblemente también para Linux: la nueva versión de la herramienta SamDisk, originalmente concebida para los discos de Sam Coupé.

Antes de pasar a repasarlas con más detalle, es de justicia nombrar una aplicación que hoy en día resulta casi imprescindible para los que trabajamos con DSKs: el SPIN Disk Manager de Damien Guard. Aunque no entraremos aquí a explicar su funcionamiento, incluimos un enlace a la página web del programa, puesto que os resultará un útil compañero a la hora de, por ejemplo, detectar posibles deficiencias en vuestros DSKs.

CPD TOOLS (CPDRead 3.24 / CPDWrite 1.03)

En los números 1 y 2 de esta revista, Miguel realizó un reportaje muy detallado sobre estas dos herramientas, creadas por Kevin Thacker (uno de los autores del formato EDSK). Dada la minuciosidad de dicho artículo, no incidiremos de nuevo sobre ello. No obstante, es conveniente remarcar que CPDRead es, de las públicamente disponibles a día de hoy, la herramienta más potente para preservar discos de 3" en formato EDSK. Por tanto, si nuestro objetivo es preservar discos de 3", será absolutamente necesario crear una partición con MS-DOS para poder sacar pleno potencial a la unidad a través de esta herramienta.

Las CPD TOOLS siguien siendo hoy en día las herramientas "oficiales" para volcado de discos, y son capaces de manejar muchas de las protecciones de disco de +3 y CPC. CPDRead es capaz de procesar discos con tamaños mixtos de sector, errores de datos, marcas de dirección (address marks) borradas y sectores de 8K truncados. Al ser una herramienta antigua, no incluye las últimas revisiones del formato EDSK, con lo que no es capaz de almacenar múltiples copias de los sectores erróneos o débiles (utilizado por el sistema Speedlock) o almacenar

más de 6144 bytes de los sectores de 8K (necesario p.e. para Coin-Op Hits, Final Fight u otros).



CPDRead en funcionamiento

Tampoco es capaz de manejar discos que tienen tamaños de cabecera ID inválidos (empleado p.e. en los discos de utilidades de Kobrahsoft SP5, SP6 y SP7). No obstante, es capaz de procesar la inmensa mayoría de discos publicados para el +3.

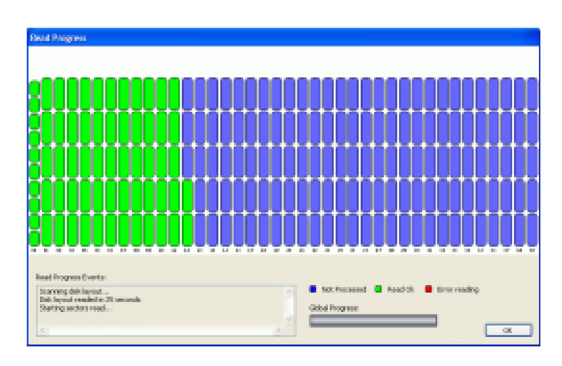
CPCDiskXP v1.6

CPCDiskXP es una herramienta desarrollada por Óscar Sánchez, alias Mochilote, para poder transferir discos a DSK, y viceversa, desde Windows 2000 o superior. Es, sin duda alguna, un excelente complemento para CPD TOOLS; y para algunas cuestiones, un más que digno sustituto. CPCDiskXP requiere instalar un driver elaborado por Simon Owen (autor de SamDisk) para acceder a bajo nivel a la controladora de disco, llamado fdrawcmd.sys, y que viene incluido en el propio paquete de CPCDiskXP (basta con lanzar un ejecutable para que quede instalado en el sistema de forma permanente, siendo desinstalable del mismo modo). A la fecha de redacción de este artículo, la última versión disponible de CPCDiskXP es la 1.6.

El funcionamiento de CPCDiskXP es muy intuitivo, gracias a su atractiva interfaz gráfica. En la pantalla principal contamos con dos grandes iconos para pasar de DSK a disco 3" y de disco 3" a DSK, y con otros botones más pequeños para poder formatear un disco u obtener información del mismo. Durante la lectura de un disco, se nos muestra un mapa de sectores en el que se puede seguir el proceso y observar el estado de cada uno de los sectores. Para la escritura, se nos muestra simplemente una barra de progreso.

CPCDiskXP soporta discos estándar de +3 y CPC y algunas protecciones de disco relativamente básicas. En la versión 1.6 aún no es capaz de manejar discos con tamaños mixtos de sector, IDs falsos, errores de datos, sectores de 8k truncados, o sectores débiles (weak sectors). Desgraciadamente, esto significa que la mayor parte de los discos protegidos no pueden ser volcados de forma correcta. No obstante, su interfaz gráfica lo hace ideal para tareas básicas. Además, nos permite elegir el número de reintentos en caso de error en lectura, algo de lo que CPDRead carecía y que sus usuarios siempre han echado en falta (aunque en caso de error final, el programa deja desafortunadamente todo el sector en blanco en el DSK).

A pesar de ello, CPCDiskXP es una herramienta muy útil en los tiempos que corren, teniendo en cuenta que nos permite hacer uso de la disquetera desde Windows de forma rápida e intuitiva, sin necesidad de abandonar nuestra sesión del Sistema Operativo.



SamDisk 2.1.1

SamDisk es una herramienta para Windows 2000/XP, originalmente pensada para la manipulación de discos de Sam Coupé, y desarrollada por Simon Owen (autor también del emulador Sim Coupé). La última versión publicada es la 2.0, pero hemos tenido acceso a una beta de la versión 2.1.1 que permitirá el manejo de discos de 3". A pesar de contar con una apariencia mucho más austera que las he-

rramientas anteriormente mencionadas (se maneja desde línea de comandos), SamDisk parece llamada a convertirse en la aplicación definitiva para el manejo de discos de 3".

SamDisk nos permite leer, grabar, formatear o examinar discos a nuestro antojo, de forma similar a cómo funciona la versión actual de SamDisk para discos de Sam. De muestra, un botón:

Copiar de disco a imagen EDSK (42 pistas, sólo cara 0):

```
SamDisk a: myimage.dsk /t:42 /s:0
```

Copiar imagen EDSK a disco (la geometría del disco se obtiene de la imagen):

```
SamDisk myimage.dsk a:
```

Examinar disco (nos da información más allá de lo que queda almacenado en EDSK, y por tanto es útil para examinar discos desconocidos):

```
SamDisk /scan a: /t:42 /s:0 /a
```

El parámetro /a muestra dónde se encontraron los sectores en cada pista, lo que permite determinar el tamaño de los huecos (gaps), así como mostrar qué sectores fueron escritos a propósito sin un campo de datos. Se puede emplear también el parámetro /scan sobre imágenes EDSK, pero en ese caso la información sobre la posición no estará disponible.

Es de esperar que Simon cambie ligeramente la sintaxis de invocación del programa en el momento en que aparezca la versión Linux, debido al uso de la barra oblicua ("/") para pasar los parámetros.

SamDisk 2.1.1 hace uso de las últimas revisiones del formato EDSK para soportar discos que anteriormente no podían ser volcados. Puede leer y escribir tamaños mixtos de sector, errores de datos, campos de datos inexistentes, sectores bobos (dummy sectors), IDs falsos y sectores de 8K truncados. Tiene la capacidad de escribir sectores débiles (necesario para los discos protegidos con Speedlock) así como sectores de más de 6K (como los de Coin-Op Hits) empleando una unidad ralentizada. Puede escribir también discos protegidos con Alkatraz, solucionando un problema con el cargador de arranque que evitaba que funcionasen en unidades de 3.5". Hasta donde se conoce, es capaz de escribir todas y cada una de las imagenes EDSK tanto de +3 como de CPC.

Por supuesto, nos permite elegir el número de reintentos en caso de error de lectura. Como puntos negativos de la herramienta, hay que destacar que los DSK que genera a veces no son compatibles con emuladores y herramientas antiguas, al seguir las últimas revisiones del formato EDSK. De hecho, al almacenar múltiples copias de los sectores erróneos, nos encontramos con que los malos volcados a menudo dan problemas con determinados programas. Como efecto lateral, el fichero DSK puede resultar notablemente más grande de lo habitual en el caso de que aparezcan múltiples sectores erróneos y hayamos marcado varios reintentos. Y, por supuesto, se echa de menos una interfaz gráfica como la de CPCDiskXP.

A pesar de esto último, podemos afirmar con rotundidad que, cuando la versión definitiva de SamDisk vea la luz, tendremos la herramienta casi definitiva para el manejo de discos de 3" en el PC. Mientras tanto, os recomendamos la instalación de un arranque dual, y el uso de CPCDiskXP para las tareas rutinarias, y CPD TOOLS para tareas de preservación y manipulación de discos protegidos más específicas.

Agradecimientos / Acknowledgments

A Jaime González Soriano, por la foto de los conectores de alimentación, y a Simon Owen por la información facilitada y la versión beta de SamDisk.

(To Jaime González Soriano, for the photo of the power supply plug, and to Simon Owen for all the information he kindly provided and the beta version of SamDisk.)

José Leandro Novellón
Juan Pablo López

Links

Guías alternativas

- Mounting a 3" disc drive on a PC: http://www.fvempel.nl/3pc.html
- Dave Batty's disk dumping guide: http://tzxvault.retrogames.com/help.htm#dsk
- Andy Barker's disk dumping guide: http://newton.sunderland.ac.uk/~specfreak/FD1/
- La guía de Deepfb: http://perso.wanadoo.es/jaimegs/ordenadores/amstrad/3enpc.htm

Artículos relacionados

- CPDTOOLS (I):
- CPDTOOLS (y II):http://magazinezx.speccy.org/index.php?revista=2&seccion=1
- Adaptador de disquetera de 3 1/2" para +3:
  http://magazinezx.speccy.org/index.php?revista=1&seccion=5

Formatos dsk/edsk: herramientas y documentación

- Ullrich Doewich's CPDRead / CPDWrite: http://www.caprice32.cybercube.com/downloads.php
- CPCDiskXP, por Mochilote: http://www.cpcmania.com/
- Simon Owen's SamDisk: http://simonowen.com/sam/samdisk/
- Damien Guard's Disk Manager: http://www.damieng.com/spectrum/spin/disk-manager/
- John Elliot's LibDsk: http://www.seasip.demon.co.uk/Unix/LibDsk/
- DU54 DSK format: http://andercheran.aiind.upv.es/~amstrad/docs/dsk.html
- EDSK format: http://andercheran.aiind.upv.es/~amstrad/docs/extdsk.html

Dual-boot de DOS y XP

- MS-DOS 7.10: http://ms-dos7.hit.bg/
- TR-Log procedure for dual-booting Windows XP and DOS:
  http://www.trlog.com/DOS _dual _boot _20041207.pdf

Proyectos de preservación de discos

- Spectrum Disk Preservation project: http://www.worldofspectrum.org/sdp/
- Spanish Spectrum Archive (preservación de discos españoles): http://spa2.speccy.org/



programación z88dk

z88dk y splib (sp1)

Ha pasado mucho tiempo desde que se publicara la anterior entrega de este curso de z88dk. Desde entonces se han puesto a disposición de los ávidos jugones algunos juegos de calidad que fueron programados utilizando esta herramienta y la librería de gráficos splib. No solo el contexto en el que nos movemos ha cambiado, sino que incluso la propia herramienta ha sufrido transformaciones relativamente profundas. Sin duda el cambio más importante es que la librería splib ha pasado a formar parte de z88dk. Eso le hace plantearse al autor de este texto si es conveniente o no continuar con aquel ejemplo del ski visto en la entrega anterior, hecho con versiones anteriores del software de desarrollo. Volver a empezar desde el principio supondría un paso atrás para aquella gente que estuviera siguiendo los ejemplos. Tan solo explicar los cambios sería muy confuso y sería poco productivo. ¿Cuál podría ser pues la solución?

Personalmente creo que para tratar esta situación, lo mejor es adoptar una posición intermedia. Vamos a desarrollar un ejemplo sencillo pero jugable, desde la instalación de la herramienta (z88dk y splib) a los toques finales. Eso sí, la intención no es hacer una explicación detallada de todos los pasos (al menos de los tratados en el presente artículo), pues al principio veremos conceptos que ya han sido tratados anteriormente. Sirva pues esta entrega de receta tanto para antiguos lectores como para novatos. Para los primeros, la ventaja será tener resumido en un solo capítulo todos los pasos necesarios para llegar al punto en el que nos encontrábamos utilizando la nueva versión de la herramienta (se puede tomar si se quiere como una receta de primeros pasos casi siempre que se quiera hacer un juego con z88dk sin scroll). Para los segundos, la ventaja es que con un poco de su parte se podrán poner al día en muy poco tiempo.

Para seguir este artículo y el siguiente se requieren conocimientos del lenguaje de programación C. No es necesario, por otra parte, tener un profundo conocimiento del hardware del Spectrum, del que nos abstraeremos en un principio (dado que no es la temática de este curso y para cubrir esa disciplina existen cursos de ensamblador en esta misma revista). Así que, sin más preámbulos, vamos a ponernos manos a la obra.

Instalación y primeros pasos

Empezamos el viaje realizando la instalación de la herramienta, que se puede encontrar en la remozada web http://www.z88dk.org. Los usuarios de Windows lo tienen fácil; en el menú de la izquierda disponen de un enlace con el título de Downloads @ z88dk.org (snapshots) desde el que descargar un binario con la última versión. Por lo tanto, no se va a hablar mucho de este sistema, y pasamos directamente a la instalación en Linux, que se hará a partir de las fuentes.

Justo debajo del enlace anterior tenemos Downloads @ Sourceforge, que nos permitirá acceder Sourceforge para descargar los fuentes. En concreto, el archivo descargado ha sido z88dk-src-1.7.tgz, que debe ser descomprimido en un directorio cualquiera de nuestro disco duro con la instrucción habitual para ello:

```
tar -xvzf z88dk-src-1.7.tgz
```

Una vez hecho esto, nos introducimos en el directorio recién desempaquetado, y para realizar una instalación válida para todos los usuarios, ejecutamos lo siguiente como root, tal como se nos indica en las instrucciones que vienen con las fuentes:

```
sh build.sh
make install
make libs
make install-libs
```

Llama la atención el hecho de que la primera vez que ejecutemos ./build.sh obtendremos un mensaje de error. Esto es así porque además de compilar las propias herramientas, antes de que éstas estén instaladas se intentan compilar los ejemplos. La solución es bien sencilla. Tras el make install volvemos a repetir el build.sh. Al instalar siguiendo estos pasos no será necesario utilizar ninguna variable de entorno.

¿Por qué sucede esto? Sencillamente, porque la primera parte del proceso de compilación con make consiste en la compilación con gcc del compilador-cruzado en sí mismo (zcc y demás binarios asociados), mientras que una segunda parte del proceso definido en el Makefile consiste en compilar las librerías de z88dk (librerías matemáticas, de stdio, splib, etc) utilizando "zcc" para ello. Obviamente, si no hemos realizado el "make install", zcc no estará en el PATH y no se podrá completar la compilación de todo el "paquete". Así pues, bastará con hacer:

```
sh build.sh
make install
sh build.sh
make libs
make install-libs
```

(También podemos usar el comando "make all" seguido de un "make install", seguido de otro "make all").

Con esto ya tendremos compilado e instalado tanto el compilador en sí mismo (zcc) como las librerías en que nos apoyaremos a la hora de programar. La ubicación habitual destino de los ficheros instalados colgará de /usr/local, como /usr/local/bin para los binarios o /usr/local/lib/z88dk para los ficheros de cabecera y librerías.

Si no queremos realizar el make install "intermedio" y queremos poder compilar todo con un solo comando (como en el listado de órdenes anterior), o bien queremos utilizar Z88DK desde un directorio concreto sin realizar la instalación, podemos definir una serie de variables entorno en nuestro "profile" de usuario del sistema:

```
export Z88DK="/opt/sromero/prog/z88dk"
export PATH="$PATH:$Z88DK/bin"
export Z80_OZFILES="$Z88DK/lib/"
export ZCCCFG="$Z88DK/lib/config/"
```

Con esto, incluiremos los binarios y librerías en nuestro PATH y no será necesario realizar el "make install" para trabajar con Z88DK. No obstante, recomendamos realizar la instalación "estándar" (make y make install) para asegurarnos de que Z88DK tiene todos los ficheros donde el compilador espera encontrarlos, especialmente para evitar posteriores problemas de compilación que pudiéramos encontrar.

Compilando SP1

La librería SP1, que utilizaremos para la creación de sprites, está incluida con z88dk pero es necesaria compilarla aparte. Para ello, desde el directorio raíz de las fuentes de z88dk, debemos entrar al directorio libsrc/sprites/software/sp1/ y teclear lo siguiente como root:

```
mv spectrum-customize.asm customize.asm
make spectrum
```

El autor de este texto tuvo que hacer una "triquiñuela" después de esto, que fue volver a ejecutar un make install general desde el raíz de las fuentes de z88dk. El procedimiento global sería:

- Compilar e instalar Z88DK y sus librerías (como se vio anteriormente, mediante build.sh y make install).
- Compilar la librería SP1 (con los 2 comandos que acabamos de ver).
- Volver al raíz de Z88DK y realizar un make install para copiar la librería y su fichero de cabecera recién compilada a /usr/local/lib/z88dk/.

Otra herramienta que nos va a resultar muy útil, además de las proporcionadas por z88dk, recibe el nombre de bin2tap, que permitirá convertir los archivos binarios obtenidos como resultado de la compilación en ficheros .tap que podremos utilizar directamente con nuestro emulador (o incluso pasar a cinta de verdad ;). Un sitio desde donde obtener el código fuente de este programa es:

http://www.speccy.org/metalbrain/

Por un extraño motivo se debe incluir la línea #include, y ya es posible compilarlo con:

```
cc -o bin2tap bin2tap.c
```

Por último, podemos mover el ejecutable obtenido a la carpeta donde hayamos dejado los ejecutables de z88dk, si así lo deseamos. Si usamos Windows, desde el enlace anterior también podemos descargar un ejecutable para este sistema operativo para poder utilizarlo sin necesidad de compilar nada.

Para comprobar que todo esté correcto, compilaremos y ejecutaremos un ejemplo. Tras introducirnos en la carpeta examples/spectrum/, tecleamos lo siguiente:

```
make gfx
bin2tap gfx.bin gfx.tap
```

Solo falta utilizar el fichero gfx.tap en un emulador y disfrutar del resultado de nuestros esfuerzos.

El "esqueleto" del juego

Tal como se ha comentado anteriormente, se va a hacer uso de la librería SP1 de z88dk (SPLIB), que ahora viene incluida en su interior y no es necesario descargar por separado, para dibujar sprites en movimiento. Más adelante hablaremos de sus características principales y novedades. Antes de eso, presentamos un esqueleto de código que deberá ser tecleado siempre que vayamos a implementar un juego que haga uso de SP1, y que deberemos rellenar con nuestras rutinas del juego. Por ejemplo, introducimos el siguiente código en un fichero llamado juego.c:

```
#include <sprites/sp1.h>
#include <malloc.h>
#include <spectrum.h>

// Dirección de inicio de la pila (0xd000)
#pragma output STACKPTR = 53248

// Declaración de la pila, necesaria para usar MALLOC.LIB
long heap;

// Definición de políticas de reserva y liberación de memoria,
// utilizadas por splib para hacer uso de este recurso
void *u_malloc(uint size)
{
   return malloc(size);
}

void u_free(void *addr)
{
   free(addr);
}

//////////////////////////////////
// INTRODUCE AQUI TUS FUNCIONES //
//////////////////////////////////

main()
{
   #asm
   // Las interrupciones son deshabilitadas totalmente al usar SP1,
   // pues esta libreria usa el registro IY, por lo que entra en
   // conflicto con las rutinas de la ROM relacionadas con este tema
   di
   #endasm
```

```
        // Inicialización de MALLOC.LIB
        // La pila está vacía
        heap = 0L;

        // Se le añade a la pila la memoria desde 40000 a 49999, inclusive
        sbrk(40000, 10000);

        /////////////////////////////////////////
        // INTRODUCE AQUÍ TU CÓDIGO PRINCIPAL  //
        ////////////////////////////////////////

}
```

¡Ojo! Es muy importante añadir un retorno de carro al final del código; en caso contrario, obtendremos un error de compilación de final de fichero inesperado.

Este programa no hace nada útil en sí mismo, es tan sólo el "esqueleto" al que añadir nuestras funciones, instrucciones, gráficos, etc. El "ejemplo" puede ser compilado y se puede crear un fichero tap de la siguiente forma (los usuarios de Linux pueden utilizar un fichero Makefile para no tener que teclear esto manualmente cada vez):

```
zcc +zx -vn juego.c -o juego.bin -lndos -lsp1 -lmalloc
bin2tap juego.bin juego.tap
```

Si ejecutamos el ejemplo anterior tal cual, el Spectrum (o el emulador) se quedará "congelado".

Para alguien que haya hecho uso de la librería de sprites anteriormente, el código anterior será totalmente diferente al que haya usado normalmente para inicializar sus programas. Para empezar, ya no se hace uso de las funciones relacionadas con IM2 (de hecho, las interrupciones se deshabilitan totalmente). Además, se hace uso de algunas funciones de malloc.h para inicializar la pila. No entraremos en mucho detalle ahora mismo. Simplemente se deberá recordar que este código deberá ser lo primero que se teclee, es decir, será el esqueleto de nuestra aplicación.

Una nueva forma de trabajar

Así pues, recapitulando lo visto hasta ahora, tenemos:

- La forma de instalar y compilar Z88DK y SPLIB.
- Un esqueleto de programa como "origen" de nuestros programas con SPLIB.
- La afirmación de que la nueva SPLIB (la integrada con Z88DK) se utiliza de forma ligeramente diferente a como se venía haciendo hasta ahora.

El lector se preguntará... ¿Realizar este cambio de librería (última versión del compilador, con SPLIB integrado en ella) merece la pena? Además de tener que cambiar totalmente los programas que hubiéramos creado con versiones anteriores de la librería, y de que la librería venga incluida en el propio paquete z88dk, hay algunas novedades muy interesantes, que pueden ser consultadas en el anuncio realizado por el propio autor en los foros de WOS:
http://www.worldofspectrum.org/forums/showthread.php?t=11729.
El objetivo de muchas de estas mejoras es incrementar la velocidad de los programas, hacer el código más limpio y más compacto, y dotar de herramientas para solucionar problemas como el color clash.

Un ejemplo de sprite

Pero volvamos a nuestra reintroducción a SPLIB y Z88DK a partir de nuestro "esqueleto de programa"

para SPLIB.

Ahora empezaremos a crear gráficos, empezando con un ejemplo sencillo. Sin embargo, es necesario primero explicar la diferencia entre los dos tipos de gráficos que puede manejar la librería.

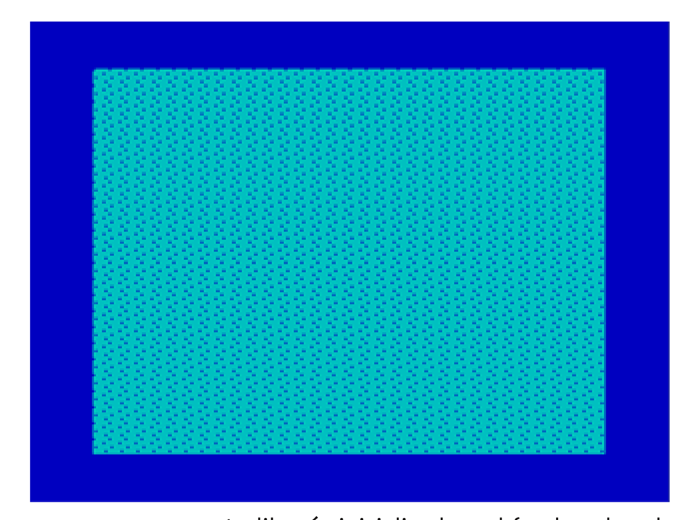
Por una parte tenemos los background tiles, que como su propio nombre se utilizarán normalmente para definir los gráficos del fondo de la pantalla sobre los que los personajes se van a desplazar.

Un tile puede ser un carácter (como por ejemplo la letra 'A') o una dirección absoluta de memoria cuyo contenido se asignará al tile. En cierta forma se pueden entender como UDGs coloreados de tamaño 8x8. La pantalla se divide en 32x24 tiles que podremos modificar. Por otra parte disponemos de sprites, entidades gráficas cuyo tamaño puede ser mayor de 8x8, y que pueden desplazarse a nivel de píxel, y que representarán los personajes, los obstáculos, los disparos, etc.

Otra característica de la librería que deberemos conocer antes de ponernos a picar código es el método que sigue la misma para dibujar en la pantalla. En lugar de actualizarla totalmente, tan solo se actualizan aquellas posiciones que han cambiado desde la última "impresión". El proceso por el cual indicamos a la librería que una posición debe ser dibujada porque se ha modificado se denomina invalidación; es decir, debemos invalidar aquellas partes de la pantalla que queremos que se redibujen.

Por ejemplo, supongamos que hacemos un juego donde se mueve por una pantalla fija un personaje que esquiva a 5 enemigos. Cada vez que dibujemos al sprite protagonista y los 5 enemigos, invalidamos las áreas donde los hemos dibujado. Esto permite que SPLIB sólo redibuje en pantalla los 6 bloques de cambios, e incluso que agrupe bloques rectangulares que se solapen (si 2 enemigos están uno encima de otro, no es necesario actualizar 2 veces esa zona de la pantalla, sino sólo una). Esto hace que la librería no pueda aplicarse a juegos con scroll, ya que implicarían invalidar toda la pantalla en cada fotograma, algo bastante costoso y que implicaría trabajar desde el principio con esa idea en mente, y dejar de lado SPLIB y cualquier otra librería de propósito general.

Y una vez dicho todo esto, vamos a empezar a teclear código, modificando el archivo juego.c que hemos creado anteriormente. El código marcado entre tags comentados "Inicio código nuevo" y "Fin código nuevo" ha sido añadido con un doble objetivo; en primer lugar se inicializa la librería, y en segundo se modifican todos los tiles de la pantalla para que tengan el aspecto que se muestra en la captura de pantalla:



La librería inicializada y el fondo colocado

```
#include <sprites/sp1.h>
#include <malloc.h>
#include <spectrum.h>

// Dirección de inicio de la pila (0xd000)
#pragma output STACKPTR = 53248

// Declaración de la pila, necesaria para usar MALLOC.LIB
long heap;

// Definición de políticas de reserva y liberación de memoria,
// utilizadas por splib para hacer uso de este recurso
void *u_malloc(uint size)
```

en segundo se modifican todos los tiles de la pantalla para que tengan el aspecto que se muestra en la captura de pantalla:

```
{
      return malloc(size);
}

void u_free(void *addr)
{
      free(addr);
}


// Definición del tile que utilizaremos de fondo en todas las posiciones
uchar fondo[] = {0x80,0x00,0x04,0x00,0x40,0x00,0x02,0x00};

// Definimos el area total en tiles (32x24 tiles) para
// poder inicializar toda la pantalla
struct sp1_Rect cr = {0, 0, 32, 24};



main()
{
      #asm
      di
      #endasm

      // Inicialización de MALLOC.LIB
      heap = 0L;
      sbrk(40000, 10000);

      //--- Inicio codigo nuevo ---

   // El borde pasa a ser azul (esto es de spectrum.h)

   zx_border(BLUE);

   // Inicializamos la libreria, y asignamos un tile a todas
   //las posiciones de la pantalla (el tile ' ')

   sp1_Initialize(SP1_IFLAG_MAKE_ROTTBL | SP1_IFLAG_OVERWRITE_TILES |
                  SP1_IFLAG_OVERWRITE_DFILE, INK_BLUE | PAPER_CYAN, ' ');

   // Ahora hacemos que el tile ' ' se represente por el patrón almacenado
   // en el array fondo. Esto ha cambiado con respecto a la versión anterior de
   // la librería, en la que esta operación se hacía antes del initialize
   // (además de tener la función un nombre totalmente diferente)

   sp1_TileEntry(' ',fondo);

   // Invalidamos toda la pantalla para que se redibuje
   // completamente en el próximo update
   sp1_Invalidate(&cr);
   sp1_UpdateNow();
```

```
    //--- Fin codigo nuevo ---
}
```

Veamos en primer lugar el significado de las declaraciones antes de la función main(). El array de unsigned char y de nombre fondo contiene una descripción del carácter 8x8 que vamos a utilizar como tile de fondo en todas las posiciones de la pantalla. Cada posición del array representa una de las 8 filas de dicho carácter, y para cada una de ellas almacenamos el valor hexadecimal correspondiente a su representación binaria (con el bit más significativo a la derecha). Viendo el siguiente gráfico se entenderá mejor cómo se ha construido ese array:

```
1 0 0 0 0 0 0 0 -> Decimal: 128  -> Hexadecimal: 80
0 0 0 0 0 0 0 0 -> Decimal: 0    -> Hexadecimal: 0
0 0 0 0 0 1 0 0 -> Decimal: 4    -> Hexadecimal: 4
0 0 0 0 0 0 0 0 -> Decimal: 0    -> Hexadecimal: 0
0 1 0 0 0 0 0 0 -> Decimal: 64   -> Hexadecimal: 40
0 0 0 0 0 0 0 0 -> Decimal: 0    -> Hexadecimal: 0
0 0 0 0 0 0 1 0 -> Decimal: 2    -> Hexadecimal: 2
0 0 0 0 0 0 0 0 -> Decimal: 0    -> Hexadecimal: 0
```

La segunda declaración antes de la función main() consiste en la creación de un rectángulo, que no es más que una estructura de tipo sp1_Rect. Los valores con los que se inicializa la variable representan la esquina superior izquierda del rectángulo (tile en la fila 0 y columna 0) y la esquina inferior derecha (fila 32 columna 24). Por lo tanto, este rectángulo representa a la pantalla completa, y se utilizará para rellenarla completamente con el tile definido en la línea anterior, como veremos más adelante.

Adéntremonos ahora en la función main en sí misma. La primera instrucción nueva es sencilla de comprender; coloreamos el borde de la pantalla de color azul mediante la función zx_border de spectrum.h. El valor BLUE es una constante definida en dicho archivo de cabecera, donde podremos encontrar constantes similares para el resto de colores.

A continuación inicializamos la librería gráfica SP1 con la instrucción sp1_Initialize. Dicha función consta de tres parámetros:

- El primer parámetro recibe como valor máscaras binarias que se pueden combinar con el operador or (|). Según el archivo de cabecera de la librería, parece ser que los tres valores que utilizamos en el ejemplo son los únicos existentes. Son parámetros no documentados (de momento) y parecen ser usados en todos los ejemplos vistos, por lo que parece conveniente dejar ese parámetro como está.
- El segundo parámetro, que también recibe como parámetro dos máscaras binarias combinadas (en esta caso solo dos), tiene una interpretación muy sencilla. Simplemente indicamos el color de fondo de papel (con una constante de tipo PAPER_COLOR, donde COLOR es el color que nosotros queramos) y de la tinta (INK_COLOR) que se utilizarán para dibujar los tiles en la pantalla.
- Por último indicamos el tile que vamos a usar de fondo.

Lo que vamos a comentar a continuación cambia un poco con respecto a la versión anterior de la librería. Antes cogíamos un carácter ASCII normal y corriente (como podría ser por ejemplo el espacio) y le asociábamos un tile de fondo (como el definido con el array fondo de nuestro código), para a continuación llamar a la función Initialize usando ese tile. Con la nueva versión, primero se hace la llamada a sp1_Initialize y a continuación se asocia al carácter el tile fondo. Esto tiene como consecuencia que para que realmente aparezca nuestro tile de fondo en toda la pantalla tenemos que ejecutar dos instrucciones más. Dichas instrucciones son sp1_invalidate() y sp1_UpdateNow() y son el fundamento de la librería SP1.

Como se ha comentado anteriormente, en lugar de andar redibujando la pantalla completa repetidamen-

te, desde el código del programa indicaremos que partes son las que han cambiado, por un proceso denominado invalidación, de tal forma que tan solo se actualizarán las partes invalidadas. Con sp1_Invalidate indicamos que parte de la pantalla queremos invalidar, y con sp1_UpdateNow forzamos a una actualización de las partes invalidadas. Como hemos tenido que asociar el tile de fondo al carácter ' ' después de inicializar la librería, nos vemos en la necesidad de invalidar toda la pantalla y actualizarla. Para invalidar toda la pantalla lo tenemos fácil. Simplemente pasamos como parámetro a sp1_Invalidate una variable de tipo rectángulo; en nuestro caso el rectángulo que definimos al inicio del programa y que cubría la pantalla completa.

Por fin tenemos el fondo de la pantalla. Es posible, por supuesto, rellenar con tiles diferentes regiones distintas de la pantalla, e incluso volcar de memoria, pero eso se verá en un futuro artículo. De momento avancemos intentando añadir un personaje en forma de Sprite que pueble nuestra solitaria pantalla azul.

Dicho sprite ocupará el tamaño de un cáracter (es decir, tendrá un tamaño de 8 filas x 8 columnas en pantalla). Con la librería libsp debemos definir cada columna del sprite por separado, entendiendo que cada columna del sprite se refiere a una columna en tamaño carácter.

Para crear el sprite del protagonista, vamos a necesitar añadir cuatro porciones de código al programa anterior:

- En primer lugar, definimos el sprite al final de nuestro código, utilizando una notación parecida a la de los tiles; es decir, el valor decimal correspondiente a la representación binaria de cada fila del sprite. Esta definición la tenemos que hacer por cada columna del sprite; en nuestro caso, tan solo definimos una columna de un carácter de alto, y referenciamos a la primera posición en memoria donde está almacenada esta columna del sprite como ._prota_colo:

```
#asm
._prota_col0

      DEFB 199,56,131,124,199,56,239,16
      DEFB 131,124,109,146,215,40,215,40

      DEFB 255,  0,255,  0,255,  0,255,  0
      DEFB 255,  0,255,  0,255,  0,255,  0

#endasm
```

Cada línea del sprite consta de dos bytes; el primero define la máscara de transparencia del sprite y el segundo el sprite en sí mismo. Como el sprite tiene 8 líneas de alto, deberemos tener 16 bytes en total (las dos primeras líneas DEFB). Además, tal como se puede apreciar, añadimos otros 16 bytes más a la columna; esto es así porque cada columna de un sprite debe tener un carácter vacío al final. Esto es necesario para que el motor de movimiento de la librería funcione correctamente (y está relacionado con la rotación del Sprite para poder moverlo píxel a píxel). Por supuesto, NO es necesario que definamos 8 bytes por línea DEFB, pero es lo que hace todo el mundo. ;)

¿Y qué carácter estamos definiendo con el código anterior? Para ello nos fijamos en el segundo byte de cada pareja de bytes. Mostramos a continuación la representación binaria de cada fila, junto con su valor decimal:

```
00111000 -> 56
01111100 -> 124
00111000 -> 56
00010000 -> 16
01111100 -> 124
10010010 -> 146
00101000 -> 40
00101000 -> 40
```

Con respecto al primer byte, con la que represen-

tamos la máscara del sprite (y con el que indicamos a través de que partes del sprite se ve el fondo), usamos los siguientes valores:

```
11000111 -> 199
10000011 -> 131
11000111 -> 199
11101111 -> 239
10000011 -> 131
01101101 -> 109
11010111 -> 215
11010111 -> 215
```

Las máscaras indican qué zonas del Sprite son transparentes y cuáles no a la hora de dibujar el Sprite sobre un fondo, de forma que nuestro sprite sea un "hombrecito" de un color y fondo determinado y no un bloque cuadrado con un hombrecito dentro.

• Añadimos las siguientes definiciones al principio del fichero, justo después de la definición de la variable heap:

```
struct personaje {
      struct sp1_ss   *sprite;
      char            dx;
      char            dy;
```

```
};

extern uchar prota_col0[];
```

La estructura personaje va a ser utilizada para modelar todos los personajes del juego, incluido el protagonista. Los campos dx y dy serán utilizados más tarde para almacenar la dirección de desplazamiento de los enemigos, mientras que la variable sprite es la que almacena la información del sprite en sí mismo. Con respecto a la variable prota_colo[], la usaremos para almacenar los bytes de la primera columna del sprite, que hemos definido a partir de la posición de memoria etiquetada como ._prota_colo. En el caso de que el sprite tuviera más columnas, deberíamos crear un array de este tipo por cada una de ellas.

• También añadimos una simple definición al comienzo del método main, en la que creamos una variable del tipo struct personaje, para el sprite que va a mover el jugador:

```
struct personaje p;
```

Por último, el código que inicializa y muestra el sprite por pantalla:

```
      // El tercer parametro es la altura (tiene que valer uno mas de lo que
      // realmente tiene, porque se deja un caracter vacio por debajo de cada
      // columna), El cuarto parametro es 0 (es el offset respecto a la
      // direccion inicial del sprite).
      // El ultimo parametro lo explicare con los enemigos
      // Los dos primeros parametros indican que el modo de dibujo será
      //con mascara, y que el sprite esta definido con dos bytes respectivamente

      p.sprite = sp1_CreateSpr(SP1_DRAW_MASK2LB, SP1_TYPE_2BYTE, 2, 0, 0);

      // El tercer parametro en la siguiente funcion es el numero de bytes
      // que ocupa el sprite, sin contar el caracter en blanco que hay que contar
      // El segundo parametro siempre tiene que ser SP1_DRAW_MASK2RB
      // para la ultima columna; las anteriores deben tener el valor
      // SP1_DRAW_MASK2.
      // Con RB se añade una columna en blanco al final, que antes era necesario y
      // ahora ya no. El ultimo parametro lo explicare con los enemigos
```

```
sp1_AddColSpr(p.sprite, SP1_DRAW_MASK2RB, 0, 16, 0);

// los valores 12 y 16 representan el caracter donde se posiciona,
// y el 0 y 0 el offset en x y en y

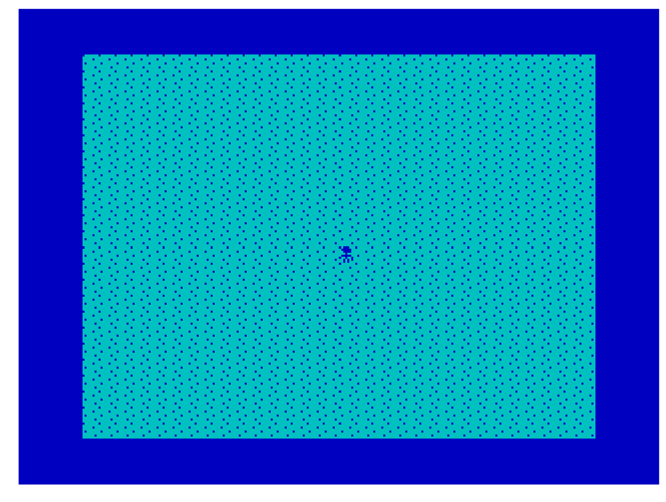
sp1_MoveSprAbs(p.sprite, &cr, prota_col0, 12, 16, 0, 0);
p.dx = 0;
p.dy = 0;

sp1_Invalidate(&cr);
sp1_UpdateNow();
```

Con sp1_CreateSpr creamos el sprite. El resultado es una variable de tipo struct sp1_ss que almacenamos en el campo sprite de la estructura que representa a nuestro personaje. Los dos primeros parámetros indican que el modo de dibujo será con máscara, y que por lo tanto el sprite está definido usando dos bytes por píxel. El siguiente parámetro, que vale 2, es la altura del sprite, en caracteres. Hemos de recordar que debemos siempre añadir un carácter en blanco al final; esto explica por qué la altura no vale 1. El cuarto parámetro, que vale 0, es el offset con respecto a la dirección inicial del sprite. El último de los parámetros lo veremos más tarde, a la hora de crear a los enemigos.

Una vez creado el sprite, debemos ir añadiendo una a una las columnas del mismo. En nuestro caso solo usamos una columna, por lo que solo haremos una llamada a sp1_AddColSpr indicando a qué sprite se le va a añadir la columna en el primer parámetro. El segundo parámetro indica el modo de dibujo, y debe valer siempre, tal cual hemos creado el sprite, SP1_DRAW_MASK2 para todas las columnas menos para la última, en la que debe valer SP1_DRAW_MASK2RB. Esto último añade una columna en blanco al final del sprite, que también es necesaria para que el movimiento funcione correctamente, y que en versiones anteriores de la librería había que introducir a mano. Como nuestro sprite solo tiene una columna, utilizamos únicamente el valor SP1_DRAW_MASK2RB. El cuarto parámetro es el número de bytes que ocupa la definición de la columna del sprite (2 bytes por fila x 8 filas = 16 bytes). No es necesario añadir el carácter en blanco del final de cada columna. El resto de parámetros se explicará más adelante.

Terminamos esta parte del código colocando el sprite en el centro de la pantalla con sp1_MoveSprAbs (el valor 12 y 16 indican fila y columna en tiles, mientras que los dos últimos valores indican el offset en número de píxels a partir de esa posición inicial), dándole al desplazamiento en x e y del protagonista el valor 0, y redibujando la pantalla. El resultado se puede observar en la siguiente captura:



Nuestro sprite preparado para moverse

Cómo mover nuestro sprite

Como viene siendo habitual, el apartado de movimiento también ha sufrido modificaciones respecto a versiones anteriores de la librería. El objetivo de este apartado es conseguir que nuestro personaje principal se pueda mover con el teclado, utilizando la combinación de teclas o, p, q, a a la que estamos tan acostumbrados los jugones de Spectrum. Para ello, es necesario ir introduciendo una serie de cambios a lo largo del código.

En primer lugar, añadimos los siguientes campos al struct personaje:

```
struct in_UDK keys;
void *getjoy;
uchar *frameoffset;
```

El campo getjoy lo utilizaremos para indicar el tipo de entrada con el que se moverá el sprite, ya sea teclado, kempston o sinclair. El objetivo de la estructura keys es el mismo que en versiones anteriores de la librería: indicar qué teclas se corresponden con cada uno de los movimientos básicos (arriba, abajo, izquierda, derecha y disparo).
Añadimos una nueva variable a las variables locales del método main:

```
uchar input;
```

Como su propio nombre indica, esta variable nos va a permitir obtener la entrada de teclado, para decidir que hacemos en funcion de ella. El siguiente cambio es un tema delicado, porque implica algo que no es realmente muy intuitivo. Anteriormente habíamos comentado que era necesario que, al definir cada una de las columnas de un sprite, añadiésemos a todas ellas un carácter en blanco al final; de esta forma conseguiríamos que el movimiento fuera correcto. Pues bien, después de hacer varias pruebas, parece ser que también es necesario añadir un carácter en blanco ANTES de nuestro sprite. En concreto, hemos añadido dos nuevas líneas DEFB antes del sprite del protagonista:

```
#asm
._prota_col0
      DEFB 255, 0, 255, 0, 255, 0, 255, 0
      DEFB 255, 0, 255, 0, 255, 0 ,255, 0
      DEFB 199,56,131,124,199,56,239,16
      DEFB 131,124,109,146,215,40,215,40
      ; Siempre tiene que haber un caracter vacio debajo
      DEFB 255,  0,255,  0,255,  0,255,  0
      DEFB 255,  0,255,  0,255,  0,255,  0
#endasm
```

Esto, evidentemente, va a producir un cambio en una de las líneas de nuestro programa, en concreto, en la que creábamos nuestro sprite, que ahora pasa a ser de la siguiente forma (el cambio es el parámetro "16"):

```
p.sprite = sp1_CreateSpr(SP1_DRAW_MASK2LB, SP1_TYPE_2BYTE, 2, 16, 0);
```

En concreto, la altura del sprite sigue siendo la misma (2 carácteres, el de nuestro sprite y el carácter en blanco al final), solo que añadimos un offset de 16 bytes, lo que quiere decir que la definición del sprite comienza 16 bytes más adelante de la dirección de memoria etiquetada como ._prota_colo.

Y por fin, al final del método main, añadimos dos nuevos trozos de código. El primero de ellos inicializa las estructuras que permitirán mover a nuestro protagonista con el teclado. Esto se hace prácticamente igual que con la versión anterior de la librería, y además es muy fácil de entender, así que no es necesario comentar nada más:

```
p.getjoy = in_JoyKeyboard;    /* Con esto decimos que el sprite se controla
      mediante teclado */
p.keys.left = in_LookupKey('o');
p.keys.right = in_LookupKey('p');
p.keys.up = in_LookupKey('q');
p.keys.down = in_LookupKey('a');
p.keys.fire = in_LookupKey('m');
p.frameoffset = (uchar *) 0;
```

La segunda porción de código conforma el bucle principal (e infinito) de nuestro programa, y permite por fin añadir el ansiado movimiento:

```
while (1)
{
	sp1_UpdateNow();

	input = (p.getjoy)(&p.keys);
	if (input & in_RIGHT && !(p.sprite->col > 30 && p.sprite->hrot > 0))
	p.dx = 1;
	if (input & in_LEFT && !(p.sprite->col < 1 && p.sprite->hrot < 1))
	p.dx = -1;
	if (input & in_DOWN && !(p.sprite->row > 22))
	p.dy = 1;
	if (input & in_UP && !(p.sprite->row < 1 && p.sprite->vrot < 129))
	// Para sprites definidos por 2 bytes, el bit 7 de vrot siempre vale 1
	p.dy = -1;

	sp1_MoveSprRel(p.sprite, &cr, p.frameoffset, 0, 0, p.dy, p.dx);
	p.dx = 0;
	p.dy = 0;

	in_Wait(5); // Para añadir retardo
}
```

La función utilizada para mover el sprite es sp1_MoveSprRel. Los parámetros interesantes son los dos últimos, que permiten mover en incrementos de un píxel en y y en x. Los dos anteriores permiten hacerlo en incrementos de tiles, pero no los vamos a usar en nuestro programa. Los valores de desplazamiento en y y en x p.dy y p.dx son inicializados en las cuatro sentencias if anteriores, en las que se determina qué teclas se han pulsado. Lo bueno de esta función es que nos ahorramos todos los cálculos a bases de divisiones y módulos que debíamos utilizar en la versión anterior para poder desplazar un sprite a nivel de píxel.

¿Y cómo sabemos qué teclas ha pulsado el usuario? Mediante la siguiente instrucción:

```
	input = (p.getjoy)(&p.keys);
```

De esta forma, almacenamos en input un byte en el que se pondrán a 1 las posiciones correspondientes a las de las direcciones cuyas teclas hayan sido pulsadas. Así pues, para conocer exactamente las teclas pulsadas, hacemos un AND lógico con unas máscaras binarias definidas como constantes en la librería, de la forma in_DIRECCION. Por ejemplo, en el siguiente ejemplo...

```
if (input & in_RIGHT)
{
   // cuerpo del if
}
```

... continuaremos la ejecución en el cuerpo del if si la tecla pulsada se corresponde con la dirección derecha. Por último, para controlar que el personaje no supere los límites de la pantalla, hacemos uso de los campos de la estructura sprite col y row, que indican la posición (expresada en tiles) del sprite, y hrot y vrot, que indican el desplazamiento en píxeles horizontal y verticalmente a partir del tile correspondiente. En el caso de vrot, hay que tener en cuenta que el septimo bit siempre valdrá 1 para sprites definidos usando dos bytes (como el nuestro), por lo que realmente los valores de vrot oscilarán entre el 128 y 135. Los valores mostrados en el código anterior significan:

• if (input & in_RIGHT && !(p.sprite?col > 30 && p.sprite?hrot > 0)):
Solo mover hacia la derecha si el tile no está situado más allá de la columna 30 con un desplazamiento en x de 0. Esto es así porque nuestro sprite solo tiene una columna de ancho. Habría que restar uno al 30

por cada columna adicional.

• if (input & in_LEFT && !(p.sprite?col < 1 && p.sprite?hrot < 1)):
Solo mover hacia la izquierda en el caso de que la columna en la que se encuentra no sea la primera, o sea la primera pero no se encuentre en el primer píxel de todos.

• if (input & in_DOWN && !(p.sprite?row > 22)):
Solo mover hacia abajo si la fila en la que se encuentra el sprite es menor de 21. Habría que restar si la altura de nuestro sprite fuera mayor.

• if (input & in_UP && !(p.sprite?row < 1 && p.sprite?vrot < 129)):
Igual que en el caso del movimiento hacia la izquierda, solo permitir mover hacia arriba si no nos encontramos en la primera fila, o si nos encontramos en la primera fila pero no en el primer píxel.

Enemigos

Vamos a añadir 6 enemigos que se moverán en línea recta por la pantalla, rebotando en los bordes cuando lleguen hasta ellos. Más tarde añadiremos detección de colisiones para terminar la partida en el caso de que alguno de ellos impacte con nuestro personaje, consiguiendo un juego muy sencillo. En primer lugar incorporamos la definición del sprite de los enemigos en la directiva #asm que se encontraba al final del código, donde también definimos el sprite del protagonista. La nueva porción de código tendrá este aspecto:

```
._enemigo_col0
      DEFB 255, 0, 255, 0, 255, 0, 255, 0
      DEFB 255, 0, 255, 0, 255, 0, 255, 0
      DEFB 231,24,195,36,129,90,0,153
      DEFB 129,90,195,36,231,24,255,0
      DEFB 255,0,255,0,255,0,255,0
      DEFB 255,0,255,0,255,0,255,0
```

Como se puede observar, se ha dejado un carácter en blanco tanto antes como después de nuestro sprite 8x8, exactamente igual que en el caso del protagonista. Esta definición se corresponde al siguiente sprite, en el que en la parte izquierda vemos el sprite en sí mismo y en la parte derecha la máscara que indica la transparencia (qué partes del sprite son transparentes):

```
00011000 -> 24              11100111 -> 231
00100100 -> 36              11000011 -> 195
01011010 -> 90              10000001 -> 129
10011001 -> 153             00000000 -> 0
01011010 -> 90              10000001 -> 129
00100100 -> 36              11000011 -> 195
00011000 -> 24              11100111 -> 231
00000000 -> 0               11111111 -> 255
```

A continuación añadimos las variables globales que necesitaremos, justo a continuación de la línea extern uchar prota_colo[]; de la siguiente forma:

```
extern uchar enemigo_col0[];
short int posiciones[] = {5,4,20,25,20,3,1,5,12,12,18,18};
short int desplazamientos[] = {1,1,-1,1,1,-1,-1,-1,-1,-1,1,1};
```

El array enemigo_colo se encargará de almacenar los bytes con los que definiremos tanto el sprite como la máscara de los enemigos. Los dos arrays definidos a continuación almacenan, para los 6 enemigos, tanto su posición inicial (en número de tile, primero la coordenada y y a continuación la coordenada x) como su desplazamiento inicial (un desplazamiento en y y otro en x para cada enemigo). Es decir, los enemigos tendrán un desplazamiento constante en los dos ejes. Cuando un enemigo colisione con un borde se cambiará el sentido de su marcha simplemente asignando a su dx o dy la inversa, según si se ha colisionado con un borde lateral, o con el superior o inferior.

En el método main añadimos las siguientes variables:

```
struct personaje enemigos[6];
int i;
```

El primer array almacenará toda la información sobre los sprites enemigos (el propio sprite, sus desplazamientos en x e y, etc, igual que en el caso del sprite protagonista). La variable i se utilizará como iterador. Ya casi estamos terminando. Creamos las seis estructuras necesarias para los sprites de los enemigos, exactamente igual que se hizo para el protagonista, de la siguiente forma:

```
for (i=0;i<6;i++)
{
      enemigos[i].sprite=sp1_CreateSpr(SP1_DRAW_MASK2LB,SP1_TYPE_2BYTE, 2, 16, 0);

      sp1_AddColSpr(enemigos[i].sprite, SP1_DRAW_MASK2RB, 0, 16, 0);

      sp1_MoveSprAbs(enemigos[i].sprite, &cr, enemigo_col0, posiciones[2*i],
posiciones[2*i+1], 0, 0);

      enemigos[i].dx = desplazamientos[2*i+1];
      enemigos[i].dy = desplazamientos[2*i];
}
```

Y por último, añadimos el movimiento de estos enemigos dentro del bucle principal del programa; es decir, justo al final del bucle infinito que comenzaba por while(1) añadimos el siguiente código:

```
for (i=0;i<6;i++)
{
      if (enemigos[i].sprite->col > 30 && enemigos[i].sprite->hrot > 0)
            enemigos[i].dx = -enemigos[i].dx;
      if (enemigos[i].sprite->row > 22)
            enemigos[i].dy = -enemigos[i].dy;

      sp1_MoveSprRel(enemigos[i].sprite, &cr, enemigos[i].frameoffset,
            0, 0, enemigos[i].dy, enemigos[i].dx);
}
```

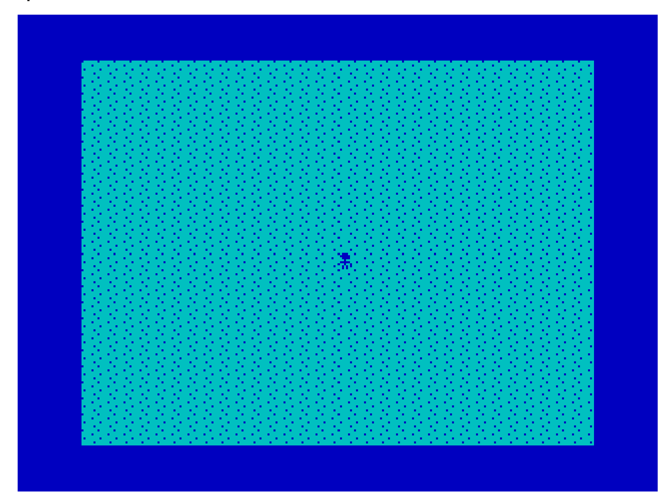
En dicho código simplemente modificamos la dirección de movimiento del enemigo correspondiente si se encuentra en los límites de la pantalla, y utilizamos sp1_MoveSprRel para realizar el movimiento en sí mismo.

¡Ya podemos ejecutar nuestro juego! Al hacerlo, veremos a nuestros enemigos desplazándose por la pantalla, mientras que nosotros podremos seguir moviéndonos con el teclado, aunque no pasará nada en especial si entramos en contacto con ellos. Sin embargo, observaremos que el movimiento se ha ralentizado muchísimo. De momento arreglamos el problema eliminando la línea in_Wait(5); que utilizábamos de retardo.

Colisiones

Solo queda un detalle para tener un pequeño juego funcional, el proceso de detección de colisiones. Lo

que haremos será permitir que el programa detecte cuando algún enemigo ha entrado en contacto con el personaje principal. Cuando esto suceda, el juego parará y se mostrará el borde de color rojo, lo que nos dará a entender que se ha terminado la partida. Tal y como se puede comprobar en diversos foros a lo largo de Internet, parece ser que es más eficiente realizar esta parte de la implementación directamente en lugar de utilizar los métodos facilitados por la librería, por lo que esta es la aproximación que vamos a utilizar.



Los enemigos hacen acto de presencia

En primer lugar introducimos las siguientes variables en el método main:

```
int xp, yp;
int xe, ye;
short int final;
```

Las variables xp e yp almacenarán las coordenadas en la pantalla del centro del personaje principal, mientras que xe e ye almacenarán las coordenadas de la esquina superior izquierda de los enemigos. La razón de esta disparidad es que al hacerlo así los cálculos serán mas sencillos. La utilidad de la variable final será tan solo la de indicar al bucle principal cuando tiene que dejar de repetirse.

El siguiente paso es sustituir la línea while(1), que se correspondía con el inicio de nuestro bucle infinito, por las dos siguientes líneas:

```
final = 0;
while (!final)
```

Con esto permitimos que cuando se detecte una colisión se le de el valor 1 a la variable final, deteniendo la ejecución del juego. Un paso más consistirá en calcular las coordenadas del centro del sprite protagonista en la pantalla una vez que ha movido, para poder comparar su posición con la de los enemigos. Esto lo haremos inmediatamente antes del bucle for que utilizábamos para mover a los enemigos:

```
xp = p.sprite->col*8 + p.sprite->hrot + 4;
yp = p.sprite->row*8 + p.sprite->vrot + 4;
```

El calculo se hace en función del tile y de la rotación. Por ejemplo, para la coordenada horizontal se multiplica la columna actual del sprite (que recordemos que está expresada en tiles) por 8, que es el número de píxeles de anchura por tile, y a continuación añadimos la rotación u offset, que nos da la posición exacta. A esto le sumamos también 4, para obtener aproximadamente la coordenada del centro del sprite (y decimos aproximadamente, porque en un sprite de 8x8 como el nuestro, el centro exacto se obtendría al sumar 3.5, pero los píxeles en pantalla son unidades discretas). Para la coordenada vertical el proceso es exactamente el mismo.

Por último añadimos el código que detecta si el sprite del protagonista entra en contacto con el sprite de alguno de los enemigos. En concreto, en el interior del bucle for que utilizamos para mover los enemigos, añadimos este código al final:

```
xe = enemigos[i].sprite->col*8 + enemigos[i].sprite->hrot;
ye = enemigos[i].sprite->row*8 + enemigos[i].sprite->vrot;
if (xe + 8  >= xp && xe <= xp && ye + 8 >= yp && ye <= yp)
{
      zx_border(RED);
      final = 1;
}
```

Una vez que se mueve el enemigo i, se obtienen las coordenadas de su esquina superior izquierda. Ahora tenemos que ver si alguna de las partes del sprite del enemigo entra en contacto con alguna de las partes del sprite del protagonista, por lo que reducimos el problema a determinar si se ha producido la intersección entre dos rectángulos de los que conocemos el centro del primero (el sprite del protagonista) y las coordenadas de la esquina superior izquierda del segundo (el sprite del enemigo). La solución a este problema se encuentra en la condición de la instrucción if mostrada en el trozo de código anterior. Si la coordenada x del centro del sprite del protagonista se encuentra entre el lado izquierdo y el lado derecho del sprite del enemigo, y además la coordenada y del centro del sprite del protagonista se encuentra entre el límite inferior y superior del sprite del enemigo, tendremos un contacto ;). Lo que hacemos en esta situación es simplemente cambiar el color del borde, y darle a la variable final el valor 1 para terminar la ejecución del bucle principal.

¡Ya tenemos un juego completo! Si, es muy simple, no hay puntos ni tabla de records, solo podemos usar unas teclas que no se pueden redefinir, pero tenemos algo que es jugable.

¿Y ahora qué?

Hay una serie de aspectos que nos dejamos en el tintero y que trataremos en próximos artículos:

- Cómo definimos sprites más grandes? Veremos como crear sprites de un tamaño superior a 8x8. En realidad es algo muy sencillo una vez que ya sabemos crear sprites de 8x8.

- ¿Cómo podemos cambiar tan solo algunos de los tiles de fondo en lugar de todos? Por ejemplo, podríamos cambiar tan solo los de una región determinada de la pantalla, o incluso leer una disposición de tiles desde memoria.

- ¿Cómo añadimos atributos? ¿Y como tratamos el color clash? Queda un poco soso el juego si solo utilizamos el color azul. Estaría bien poder cambiar el color de los sprites del protagonista y de los enemigos para distinguirlos mejor.

- ¿Cómo podemos utilizar otros métodos de control aparte del teclado? Muy sencillo también, tan solo es cuestión de conocer los posibles valores a utilizar a la hora de inicializar la estructura de entrada de un sprite.

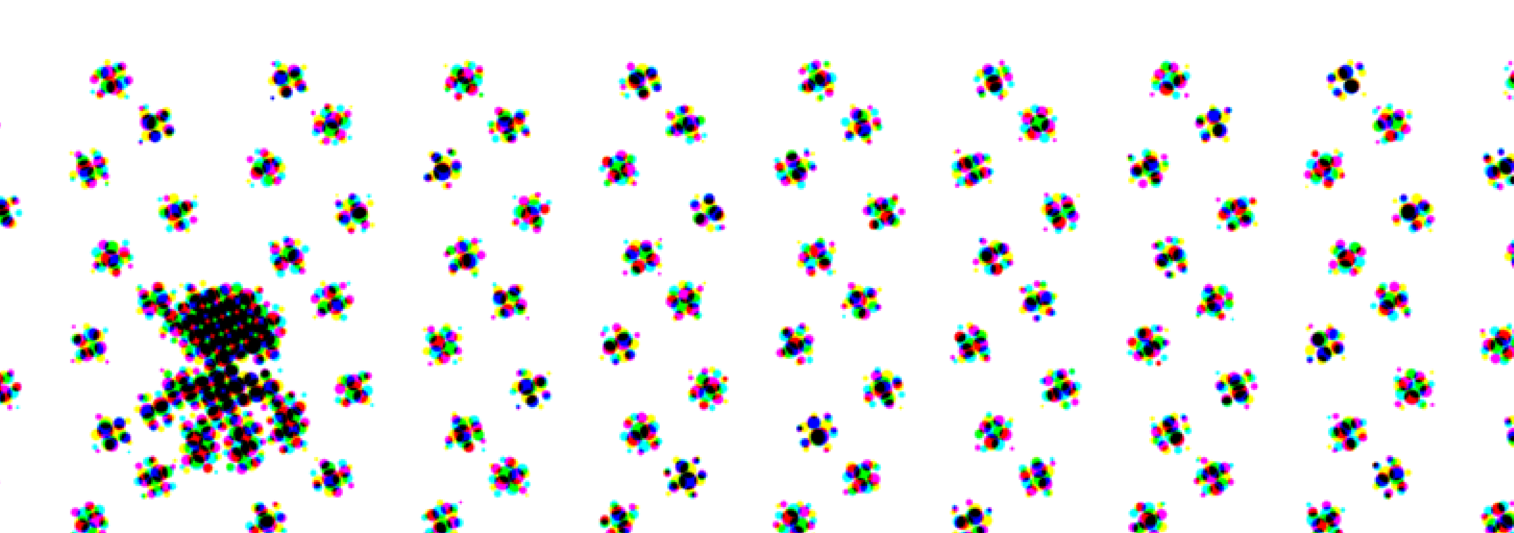
- ¿Cómo creamos enemigos con un poquito de inteligencia? Esto puede dar pie a artículos y artículos sobre Inteligencia Artificial en el Spectrum, en el que los recursos limitados supondrán un reto también en este campo.

De todas formas, gracias a este artículo se disponen de las bases necesarias no solo para programar ejemplos tan sencillos como el que se presenta, sino que incluso para profundizar en las cabeceras de la librería e intentar llegar más allá.

Links

Código fuente:
http://magazinezx.speccy.org/revistas/16/src/z88dk16_codigo.zip

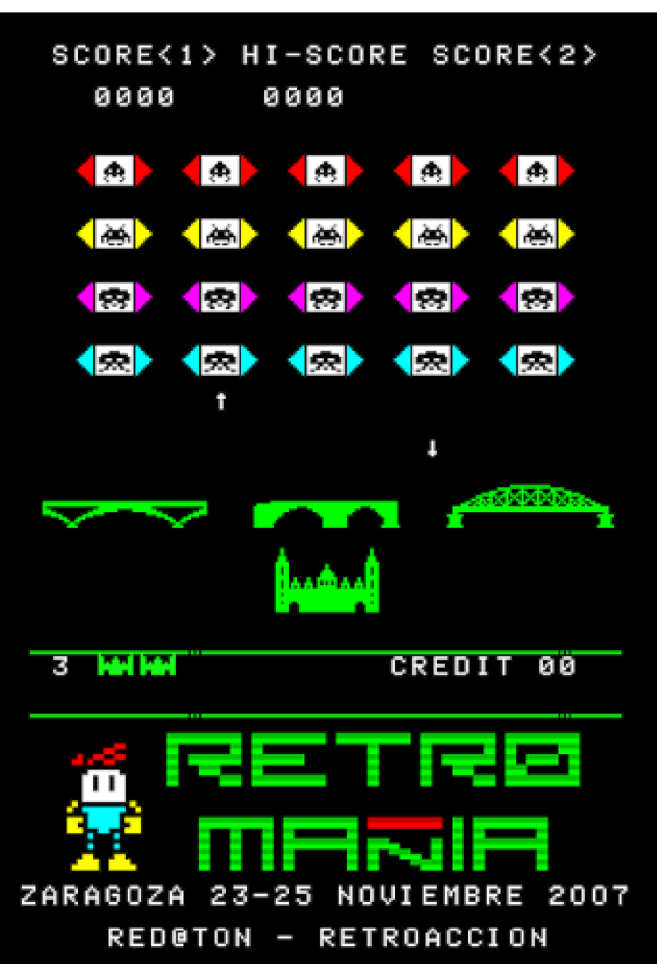
Siew





rem
retroeuskal '07 - retromañía





En esta ocasión vamos a conocer las tripas de dos eventos retro celebrados este pasado año en España. Uno de cierta tradición, como es RetroEuskal, y otro de nuevo cuño, Retromañía.



retroeuskal '07

y ya van cuatro...

RetroEuskal, camino de su madurez, es una fecha fija en el calendario anual de eventos dedicados a la retroinformática. Como es habitual, se encuadra dentro de la celebración de la Euskal Encounter en Bilbao. En esta ocasión las fechas fueron del 20 al 23 de Julio en el BEC! de Barakaldo. Como dato anecdótico, por primera vez su organización corrió a cargo de la Asociación RetroAcción.

Desde los comienzos de RetroEuskal, cada edición se ha dotado de múltiples contenidos y actividades, fijando especial atención en un determinado tema que hile la mayor parte de ellas. En 2007 se optó por dar relevancia al 25 aniversario de la salida del primer Spectrum, uno de los ordenadores que hicieron despegar el mercado doméstico informático europeo. Speccy.org colaboró económicamente patrocinando el evento, lo que supuso poder tener más recursos para el museo sobre Sinclair.

Los asiduos al lugar ya conocerán la idiosincrasia que caracteriza el desarrollo de las zonas y actividades propuestas por la organización. Sin embargo, haremos un pequeño resumen para los noveles. El museo es la parte central del espacio sobre el que gira la dinámica del resto. En esta edición, a un lado quedaba la zona de proyectos; delante, el juegódromo para aquellos que quisieran retar sus habilidades contra videojuegos clásicos; y al otro extremo, la "Morgue", lugar donde el "doctor Retro" y sus secuaces procuraron devolver a la vida máquinas averiadas. Por último, los talleres y charlas se realizaron en el espacio habilitado por Mod-PC dentro del bexitec, zona comercial de la Euskal.

El museo

Gracias a la colaboración desinteresada de varias personas, RetroEuskal contó con la que posiblemente sea la muestra pública más completa sobre Sinclair en nuestro país. Los visitantes pudieron contemplar publicaciones, productos, folletos, y demás parafernalia relacionada con la marca y con el hombre, acompañada de textos explicativos que guiaban de forma cronológica lo allí expuesto.

Tras una presentación biográfica del por entonces Clive sin Sir, el recorrido comenzaba con una serie de publicaciones de su autoría junto al número de la revista de la asociación Mensa donde fue portada. Como complemento, una foto autografiada de las que la gente mataría en ebay al grito de "L@@K, R4R3!".

La siguiente parada era la visita a Sinclair Radionics. Escaparate de la obsesión por la miniaturización, estuvieron presentes radios, calculado-


Vitrinas de Sinclair Radionics y Sinclair Research

ras setenteras con aspiraciones de altura, multímetros y amplificadores varios. Testigo de este desfile fue la Microvision.

El salto a la Research se adivinaba fácilmente con la presencia de los ZX80 y ZX81 acompañados de sus correspondencias norteamericanas y ampliaciones de la exigua memoria de estas máquinas. Para muchos, aquí comenzaba el terreno conocido, y el inicio de sus infancias felices.

Así, varios pasos consecutivos, y el espectador podía cruzar mirada con tres teclados negros con un arco iris en común. Hablamos de la vitrina que reunió a la familia Sinclair Spectrum "legítima". El pequeño 16K acompañado del Plus que aguantó el tirón mientras llegaba el anhelado y tardío 128.

Contiguo a este pequeño tesoro de plástico se encontraba una ínfima muestra de las expansiones que acompañaron a esta gama de ordenadores, desde interfaces para conectar en red, pasando por mandos para ordenadores en un principio "serios", microdrives sin fin con final previsible o impresoras de papel continuo con escatológica analogía.

El fin de la producción de la Sinclair original vino de la mano del QL, olvidado tras la absorción por parte de Amstrad PLC. La exposición inconscientemente lo dejaba rodeado por un +2, un +2A y un +3. Un ordenador con miras profesionales arrinconado por videoconsolas con teclas. Aunque no eran de Clive, pensándolo bien, el concepto no deja de ser absurdo.

Estas máquinas necesitan de programas que las doten de utilidad, por lo que no podía faltar una muestra heterogénea de software. La propia Sinclair Research se encargó inicialmente de nutrir a sus ordenadores de programas educativos, profesionales y lúdicos, con una imagen corporativa muy familiar para los viejos del lugar. La producción externa optó por crear sus propios diseños, generándose una obscena cantidad de formas y colores en las presentaciones del producto, que hoy en día extrañan ante la uniformidad de las estanterías de los comercios de software.

Otra vitrina más y el visitante tropezaba con la prensa escrita especializada. Unos cuantos números uno de revistas españolas, inglesas y estadounidenses que recogían el devenir de los ordenadores Sinclair en todas sus facetas: programación, hardware, software,...

Todavía quedaba espacio para hallar una formación de clones de procedencias bien distintas y del nuevo proyecto bajo el sello Cambridge computers. Timex, Investronica y Miles Gordon Technology hicieron diferente fortuna aprovechando el tirón de Sinclair, mientras que el Z88 se convertiría en el último ordenador relacionado con el hombre del que aquí hablamos.

El final de la exposición, como no podía ser de otra forma, venía marcada por un práctico reloj radio, una televisión de "pantalla plana" y el leaflet de un vehículo eléctrico descapotable ideal para el lluvioso clima británico. Algunos hablan de despropósito. Otros, de visión de futuro. Pero demasiado lejana como para comercializar un producto bastantes años antes de un cambio climático y una carestía de combustibles fósiles.

Como veis, material de sobras como para escribir un libro, tal como pensó Rodney Dale.

Rick Dickinson

A principios del mes de Julio, en los foros de "The World of Spectrum" asistimos al anuncio de una serie de galerías de fotos con material inédito o poco conocido de la Sinclair Research. Se trataba del propio Rick Dickinson, diseñador principal de la compañía hasta la absorción de Amstrad PLC. Por medio de Albert Valls, se estableció un puente de comunicación del que salió un buen puñado de frases muy esclarecedoras sobre aquella época. El resultado desembocó en una serie de láminas que acompañaron al museo, donde por medio de las fotos y comen-

La llegada de Amstrad, publicaciones y clones

tarios del propio Dickinson, se ponían a la luz datos sobre los desarrollos de la empresa, así como explicaciones sobre los prototipos que todos conocen y nadie llegó a ver: Loki, Janus, Pandora...

Proyectos

Speccy.org no necesita más presentación en esta revista. Su presencia en RetroEuskal sirvió para dar a conocer las webs y servicios que ofrece a la comunidad de aficionados de Sinclair. Por ejemplo, Magazine ZX estuvo en formato papel a través de los últimos números publicados.

Por otra parte, y empieza a ser tradición, Jörn Mika (Prodatron) acudió a Bilbao puntualmente para mostrar los avances de su sistema operativo multiplataforma Symbos. Ya sabemos que funciona en los CPC y MSX-2 (ahora también en los PCW). Las novedades de este año se centraron en el apartado multimedia, con la compatibilidad con la tarjeta descompresora de mp3 para los MSX y un navegador en desarrollo. Esta última aplicación trabajará en conjunción con la tarjeta ethernet para CPC diseñada por Matthias Hüsch (Dr. Zed), quien también estuvo presente en el BEC.

Vista general y en detalle

Talleres

Se desarrollaron durante el sábado y domingo, con una temática bastante amplia.

Aparte del espacio fijo asignado durante todo el fin de semana en la zona de proyectos, Jörn y Matthias también hicieron presentación pública de sus avances con Symbos y nuevo hardware en sendas charlas. Por desgracia, la tarjeta ethernet prototipo para CPC sufrió daños en el viaje que impidieron la demostración práctica, aunque el Dr. Zed no tuvo problemas para explicar sus funcionalidades e interioridades.

Previamente a su intervención, en la mañana del sábado, Eduardo Mena, profesor del Departamento de Informática e Ingeniería de Sistemas de la Universidad de Zaragoza, expuso la variedad de opciones que tenemos para conectar las salidas de audio y vídeo de los sistemas de 8 bits. Se hizo un amplio repaso a como mejorar la exigua calidad de las tradicionales tomas de antena que se tuvieron por estándar durante los ochenta.

En la mañana del domingo Rafael Corrales y Miguel Durán, presidente y secretario de la Asociación MadriSX.org (ahora AUIC), presentaron la feria que se organiza en Madrid cada año

Sinclairmanía y "confesiones" de Rick Dickinson

y los objetivos de su proyecto.

El cierre de las charlas lo dio por la tarde Antonio Jesús Rodriguez, haciendo un amplio repaso a la historia de la Informática doméstica, acompañando su exposición de abundantes fotografías de equipos clásicos, informaciones técnicas y anécdotas de los mismos.

MicroHobby: Un micro, un hobby, una revista

Con este lema se presentaba el plato fuerte para muchos. Domingo Gómez, director y editor de esta revista y de Micromanía, y Primitivo de Francisco, redactor experto en hardware, acudieron a la mesa redonda que culminaba el homenaje-celebración.

La charla estuvo dirigida por Albert Valls. Las dos horas de duración se hicieron cortas, pero sirvieron para conocer de primera mano como surgió el proyecto, los inicios, la competencia... En el campo de las secciones de la revista se habló sobre los concursos realizados, los proyectos de hardware, los listados de programas junto

Los talleres y proyectos de RetroEuskal



a la aparición del Cargador Universal de Código Máquina, los justicieros del software o un hipotético trato de favor al software español. Otras cuestiones surgidas fueron la polémica que podía suponer los artículos de la biblia del hacker frente a la defensa del software original contra la piratería, o a modo de anécdota, si existía realmente Alan Heap.

Sin embargo, la mesa redonda no se quedó sólo en palabras. Primitivo de Francisco tenía reservada una pequeña sorpresa. Tuvo la deferencia de mostrar a los asistentes el Spectrum modificado que utilizaba para sus investigaciones y montajes de hardware. La máquina se compone de una placa de un 48K dentro de una caja metálica con teclado y conectores de expansión profesionales. Incorpora cassette, varios años antes que los modelos de Amstrad, minijoystick, cursores modificados, reset, botón NMI, salida de vídeo compuesto, disipación de calor y mejoras en la toma de sonido del cassette. Su aparición en escena provocó tumultos y flashes de cámara cual consola de nueva generación presentada en el E3 estadounidense. Sobre todo, gracias a la predisposición de Primitivo a mostrar las interioridades del "Súper Spectrum".





Primitivo y Domingo en la conferencia. En medio, el invitado sorpresa

El juegódromo y las nuevas disciplinas retrodeportivas

El público asistente a RetroEuskal no se encuentra sólo con máquinas de exposición. Tiene a su alcance una muestra heterogénea de sistemas que trata de cubrir los más comunes en la época, así como otros más anecdóticos. Todo dispuesto para poder degustar juegos clásicos.

Los invitados fijos de rigor de los 8 bits, Spectrum, Amstrad, MSX o Commodore, contaron con compañeros como los Atari St, Amiga, e incluso alguna consola como Master System. Por primera vez se dispuso de una máquina arcade con la colaboración de MOD-PC. Aunque, al fin y al cabo, fuera un sucedáneo sustentado por el prestigioso proyecto M.A.M.E., sirvió de continuación al tímido inicio de la presencia de recreativas en la edición anterior. Esperemos que sea un acicate más para aumentar la presencia de este sector del entretenimiento digital.

Varios desarrolladores de software nuevo para máquinas de 8 bits tuvieron la deferencia de estar presentes con algunos de sus juegos. Computer EmuZone Games Studio, CNG soft,

El Dr. Retro, una panda de frikis y los ganadores de las Retrolimpiadas



Cronosoft y Compiler software tuvieron una máquina dedicada para que los visitantes pudieran conocer parte del movimiento de ocio actual que generan estos microordenadores.

Con las Retrolimpiadas se completó todavía más la oferta de actividades para el asistente. A través de pruebas un tanto bizarras, se buscó al usuario obsoleto más competente: rebobinado de cinta con bolígrafo Bic, partida al Bomb jack con joystick al revés, lanzamiento de disco de 5" 1/4, adivinar el color que muestra una pantalla en blanco y negro... El objetivo de pasarlo lo mejor posible se cumplió de sobras, aunque no dejó de haber una sana competencia entre los participantes, y sorpresas en cuanto a la (poca) habilidad de alguno de ellos. Los ganadores obtuvieron su diploma correspondiente acompañado de un pequeño obsequio.

La morgue del doctor Retro

Bajo esta denominación tan tétrica y teatral en realidad se esconde el taller de hardware de RetroEuskal: Jaime Lapeña (Rockriver), Mikel Erauskin (KaosOverride), David Gonzalo (z80user) ... Hablamos de gente dispuesta a investigar en las entrañas de equipos obsoletos que no han resistido el paso de los tiempos y duermen el sueño de los justos. Es el lugar ideal para intercambiar información que ayude a reparar, o incluso ampliar sistemas de otro siglo, como ya han podido comprobar los visitantes en las últimas ediciones.

Demoscene envasada al vacío

O más correctamente, lo que quedó vacío fue la convocatoria que se realizó para la competición de producciones. Posiblemente, esto sea debido a la escasa tradición de nuestro país en cuanto a organizar compos para máquinas de 8 bits. Sin embargo, se procurará insistir en futuras ediciones. Queremos ver exprimir las capacidades técnicas de estos circuitos y la creatividad de los programadores, grafistas y músicos de la escena.

Por suerte, KaosOverride no faltó a la cita con la pantalla gigante de la Euskal, a pesar de alguna que otra discusión con un disco duro rebelde. La hora de los 8 bits hizo un repaso por las últimas producciones de sistemas varios, sorprendiendo de nuevo al personal con las ocurrencias surgidas de sinapsis pixeladas.

Tres días que son seis

Cinco palabras, para intentar resumir el montaje, desarrollo y recogida de RetroEuskal. Seis meses para intercambios de sugerencias, opiniones, actividades, correos, llamadas telefónicas, acuerdos, desencuentros, organización... Cuatro o cinco horas de sueño bilbaíno por día (con suerte) para que todo funcione. Cifras que desembocan en otra más: doce meses que empiezan en un lunes gris y que marcan una distancia y horizonte bien definido: RetroEuskal 2008. ¿Acaso te lo vas a perder?

Links

RetroEuskal: http://www.retroeuskal.org/
RetroAcción: http://www.retroaccion.org/
Euskal encounter:
http://www.euskalencounter.org/
Akton films: http://aktonfilms.speccy.org/

La puerta de Sinclair

Gracias a Paul Ruiz y Josetxu Malanda se proyectó el cortometraje recientemente estrenado sobre una dimensión alternativa inspirada en el "universo Sinclair". Ellos mismos nos cuentan como fue el desarrollo del proyecto

Sir Clive Sinclair fue un genio que tuvo la desdicha de perderlo casi todo a mediados de los años 80 por una serie de decisiones comerciales desafortunadas. Pero, ¿qué habría sucedido si Sinclair no hubiera fracasado?

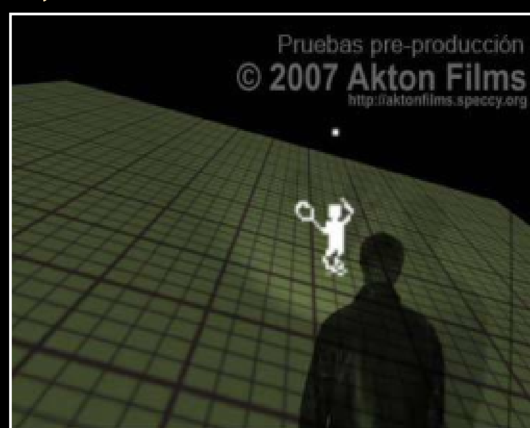
Bajo esta premisa hace ya varios años nos propusimos desarrollar una historia para rodar un cortometraje, pero no fue hasta el año 2006 cuando comenzamos a tomarlo realmente en serio. Una cosa estaba clara, el corto tenía que homenajear a los grandes juegos del ZX Spectrum y para ello nada mejor que sus escenas finales transcurrieran en un entorno basado enteramente en juegos de Spectrum. Así que lo primero era buscar el "look" de estas escenas.

Lo primero que vino a mi mente fue utilizar la croma para introducirnos directamente en las imagenes de los juegos, respetando la perspectiva lateral típica de los primeros juegos. Así que realizo una primera prueba y queda tal que así:



Ante ustedes la primera prueba para este corto

El resultado como salta a la vista es poco menos que horrendo. Pero jugueteando con el entorno 3D del estupendo programa After Effects parece que la imagen queda mucho más real e integrada. Una rápida prueba haciendo croma directamente contra una pared blanca me deja mucho mejor sabor de boca:



Otra prueba, en el entorno 3D del After Effects

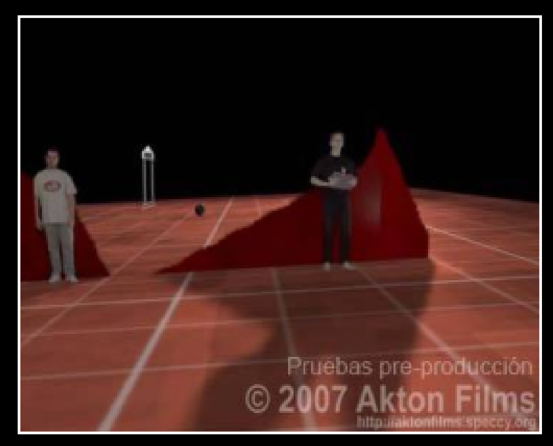
Perfecto, ya tenemos el look. Ahora solo nos falta todo lo demás, empezando por el guión.

La Preproducción

Unos metros de tela azul...

Tras comprar dos bloques de tela azul comienzo a pensar las escenas de croma donde transcurrirán las escenas finales del corto. La idea inicial de interpretar el corto yo mismo junto con Josetxu rápidamente se diluye. Está claro que la actuación no es lo nuestro, a juzgar por nuestras primeras

pruebas. Eso sí, las escenas de "efectos especiales" van quedando muy bien.



Dos pedazo de actores, solos ante el peligro

Además de nuestra inutilidad como actores, también queda patente que mi actual cámara de vídeo se queda bastante corta en cuanto a calidad de imagen, así que me agencio una Panasonic NV-GS400 de segunda mano, que ha resultado ser una excepcional compra.

Tras varios borradores y después de descartar varias ideas inviables por nuestra alarmante falta de presupuesto, en Julio de 2006 termino de escribir el guión. Tras una visita a un taller de teatro la idea gusta y consigo los tres actores que necesito.

Conseguir ZX Spectrum, cassettes, revistas Microhobby, pistolas Sinclair y demás atrezzo de le época no resultó ningún problema gracias a que tanto Josetxu como mi tío cedieron parte de su colección.

El Rodaje

Unos metros de tela azul...

El rodaje se efectuó los días 6, 7, 9 y 11 de Septiembre de 2006. El día 9 rodamos en exteriores, concretamente en el aula de Cultura de Getxo que muy amablemente nos cedieron una sala completa para nosotros solos durante todo el día.



El cámara jugándose el tipo

Los días 6,7 y 11 grabamos en nuestro sencillísimo plató de pantalla azul, situado en la cocina de mi casa. Desgraciadamente esos días fueron particularmente calurosos, lo que unido a los 3 focos halógenos que usábamos, convertían la cocina en una auténtica sauna.



Los sufridos actores frente a la casera pantalla azul

La postproducción

Unos pocos retoques por aquí...

A mediados de septiembre y con todas las escenas ya rodadas comienza oficialmente la postproducción. Hay que incorporar las escenas de croma en los escenarios que previamente había creado, crear la música, los efectos de sonido y los efectos especiales.

Especialmente lenta es la realización de los planos de las escenas finales, en las que todo está generado por ordenador a excepción de los actores. Estas escenas requieren de un trabajo ingente al tener multitud de efectos (croma, luces y sombras virtuales, explosiones, rayos y demás) Unido a que los cálculos se realizan sobre un viejo AMD XP 2000+ hace que los tiempos de render sean eternos.

En paralelo a este trabajo, nuestro músico Aritz Villodas va desarrollando la música conforme le voy pasando las escenas ya terminadas.

Ahora, tras varios meses de trabajo, el proyecto ha sido finalizado y todo este esfuerzo se ha convertido en un cortometraje de ciencia ficción con bastante acción y unas gotas de terror, plagado de homenajes a los grandes juegos de primerísima época.



Los efectos visuales transportan a los actores...

Este corto lo podéis disfrutar de manera online (con subtítulos en inglés) y totalmente gratuita en:

http://video.google.es/videoplay?docid=-7054604605367370920&subtitle=on&pr=goog-sl

Esperamos que lo disfrutes y no olvides dejarnos tu comentario (bueno o malo), visitando nuestra página web en: http://aktonfilms.speccy.org

Bienvenido al mundo de Akton Films. Comprueba con tus propios ojos qué habría sucedido si Sinclair no hubiera fracasado.





retromañía

Por primera vez Zaragoza cuenta con un evento dedicado a la retroinformática. Tuvo lugar dentro de la LANparty Red@ton, entre el 23 y 25 de Noviembre en la Sala Multiusos del Auditorio. La asociación RetroAcción se propuso colaborar con la Fundación CESTE para aprovechar la posibilidad de mostrar las múltiples caras de nuestra afición a público en general. La característica más llamativa de la Red@ton es el disponer de una zona abierta. En ella, diferentes asociaciones y marcas comerciales pueden ofrecer actividades y productos relacionados con las Nuevas Tecnologías (o no tanto) a los visitantes. A modo de ejemplo, Simaragón hizo un amplio despliegue de equipos con simuladores de vuelo, incluida cabina de avioneta, y Arashi dispuso una muestra de consolas de nueva generación en conjunción con variedad de videojuegos musicales: Guitar Hero, Beatmania, Stepmania, Taiko no Tatsujin,...

15 días por delante y sin nada preparado

Como reza el título, la primera reunión para organizar la zona abierta se realizó dos semanas



antes de la inauguración. De hecho, ni siquiera se tenía pensado un nombre para la sección obsoleta. Para solucionar el tema de la identidad, nada mejor que recurrir a tópicos. Siempre funcionan y la intertextualidad hace el resto. Así, usando conceptos manidos y representativos de la ciudad y de la afición, se llegó a la denominación de RetroMañía. El mañobot con su cachirulo de píxeles cerraban el círculo y vengaban la afrenta que supuso Fluvi. Sólo quedaba un cartel que lo representara todo. No hubo más que seguir la estela, y mezclar en la marmita marcianos envolviendo adoquines del Pilar que atacan ferozmente al corazón de Zaragoza. Lástima que no supieran como se las gasta cuando está sitiada...

Tenemos envoltorio. ¿de qué lo rellenamos?

Una vez que sabíamos quienes eramos, había que plantearse qué queríamos ser. Para la primera vez que la ciudad contaba con un evento de estas características, se consideró oportuno optar por hacer un repaso a los primeros ordenadores que entraron a los hogares de

forma masiva en los años 80.

Como punta de lanza, se estableció la zona interactiva, con equipos dispuestos para que cualquiera pudiera acceder al manejo de ordenadores y consolas. Los cuatro jinetes de los 8 bits hicieron acto de presencia: Spectrum, Amstrad, MSX y Commodore. Los 16 bits fueron representados por un Amiga 500 (aunque finalmente no pudiera lucir) y los Atari ST. Por último, y no menos importantes, también se pudo probar una Atari 2600 Jr y una Colecovision. La idea era dar la posibilidad de recordar viejos tiempos a los que habían tenido estos aparatos en sus casas, pero también mostrarlos a las nuevas generaciones. Para nosotros fue una satisfacción el ver a padres con sus hijos explicando el contraste con las máquinas que hoy en día tenemos en casa, o niños de las visitas escolares preguntando inocentemente si una Super Nintendo era una consola muy antigua.

Es complicado que personas no conocedoras de los sistemas clásicos se acerquen a su manejo, por lo que la mejor forma de franquear la barrera es a través de los videojuegos. Quién sabe si en futuras ediciones se puede conseguir que programen un poco de ensamblador de Z80... Por nosotros no quedará. Retomando la idea del inicio del párrafo, se dispuso una buena cantidad de juegos clásicos. Uno de los más exitosos fue Midi Maze, en los Atari ST, gracias al juego por red a través del interface MIDI. Muchos fueron los que compitieron por exterminar al adversario. ¿Quién puede resistirse cuándo tu rival te sonríe a punto de morir?

Los que ya tienen sabido de memoria el catálogo de videojuegos de la época, tuvieron la posibilidad de conocer alguno de los próximos lanzamientos de Computer EmuZone Game Studio. Angel Ló (Anjuel) estuvo representando al grupo, y gracias a él se pudieron probar versiones beta de alguno de los próximos lanzamientos para Spectrum:

- UWOL, programado por na_th_an: un plataformas con recogida de objetos ante el acecho de decenas de enemigos repartidos por multitud de niveles.
- Ilogicall, programado por Benway: la adaptación de los nonogramas del Picross que triunfa en Nintendo DS.
- I need speed, programado por Metalbrain: juego de carreras de coches a toda velocidad.

Aún queda sitio para "Los otros"

Se consideró interesante incorporar un pequeño

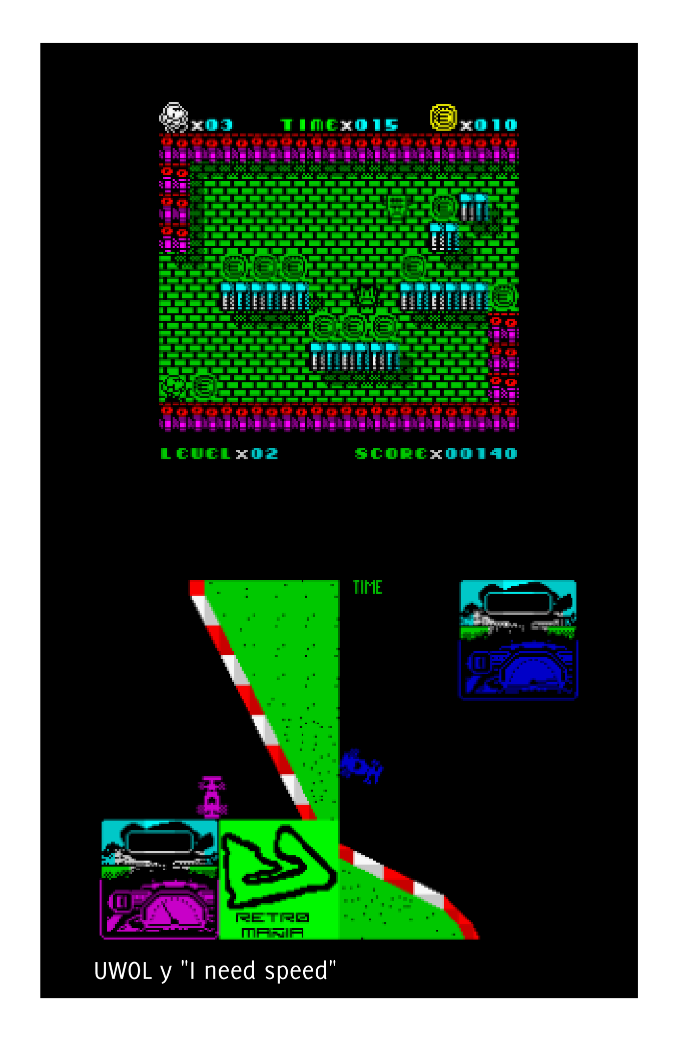


José Leandro comprobando la memoria de un "gomas"

espacio destinado a exponer equipos que son menos habituales de ver en nuestro país. La intención era que los visitantes pudieran hacerse idea de la cantidad de máquinas que inundaron las tiendas de electrónica en la década de los 80. Debido a diferentes circunstancias, muchos sistemas tuvieron una expansión testimonial: bajas ventas, mala política publicitaria, distribución deficitaria, nulo apoyo de las casas de software, diseños anacrónicos... Así se reunieron un Atari 800 con su tableta digitalizadora, un Oric Atmos, un Sharp MZ-700, un Exelvision EXL 100 con sus curiosos mandos inalámbricos,un MSX 2+, la GX4000 con un CPC 464 plus de Amstrad y una consola Pong clónica. Una muestra heterogénea que nos llevaba a reflexionar en como por ejemplo, equipos basados en un mismo procesador Z80 eran incompatibles, y en contraste, hoy, PCs con procesadores completamente diferentes son perfectamente compatibles. En sólo treinta años, los cambios han sido radicales.

Diga 100001

Aprovechando la visita de José Leandro Novellón y la colaboración de Jaime Lapeña (Rockriver) se



UWOL y "I need speed"

Salvador y Eduardo mostrando el One Chip MSX



El adaptador de mandos de PS2 para Spectrum tiene múltiples usos

ofreció un servicio de reparación de ordenadores clásicos. En cuanto a los Spectrum, se actualizaron eproms del proyecto plus3e y se arreglaron varios 48K con diferentes síntomas. También se tuvo la oportunidad de consultar a Eduardo Falces sobre cuestiones de hardware con los MSX. Como anécdota, un visitante hizo uso del servicio para tratar de arreglar un problema de reconocimiento de un pendrive en su PC. No se consiguió averiguar si el dispositivo de almacenamiento era para un 48K o 128K, por lo que se le recomendó que reinstalara el Basic XP.

Oiga, todo esto ya lo tengo muy visto...

El aficionado a la retroinformática posiblemente ya conozca todas estas máquinas, por lo que es interesante ofrecerle talleres que muestren novedades o aspectos de otros sistemas que no tenga tan revisitados. Por ello, RetroMañía programó tres charlas, adentrándose en esta ocasión en Spectrum y MSX:

- Composición de música en el MSX: el viernes por la tarde Salvador Perugorría (Xenon) hizo un preciso repaso a todas las posibilidades sonoras del estándar japonés, con abundantes ejemplos

a través de programas de composición y videojuegos tanto comerciales como amateur: sonido PSG, OPL, FMPAC,el chip de sonido SCC de Konami, la tarjeta de sonido Moonsound,...

- Ampliando las capacidades de un Spectrum: nada mejor que empezar la mañana del sábado con José Leandro Novellón haciendo una presentación de varios de sus últimos proyectos. Hizo un repaso a los diferentes dispositivos de almacenamiento con tarjeta Compact flash, enseñando una implementación de Pera Putnik para los modelos de 48K. También hizo una demostración práctica de la placa de diagnóstico de averías y el adaptador de pads de Playstation 2 a Spectrum, el cual, por cierto, es bastante versatil.

- Presentación del One Chip MSX: pocos son los afortunados que han podido hacerse con uno de estos nuevos MSX. Comercializado por la empresa japonesa D4 enterprise, se basa en una FPGA Altera Cyclone. Da soporte completo al MSX2, sonido PSG, SCC+ y MSX-MUSIC (FM-PAC). Además cuenta con conexión PS/2, 2 puertos USB, 2 puertos de joystick, 2 slots de cartucho y uno de tarjetas SD/MMC. La salida de vídeo es a través de VGA, video compuesto y S-Video, teniendo dos





Nuestros cuatro ponentes

salidas de audio. De nuevo Salvador Perugorría, acompañado de Eduardo Falces, mostraron todas las capacidades de esta pequeña caja azul translúcida.

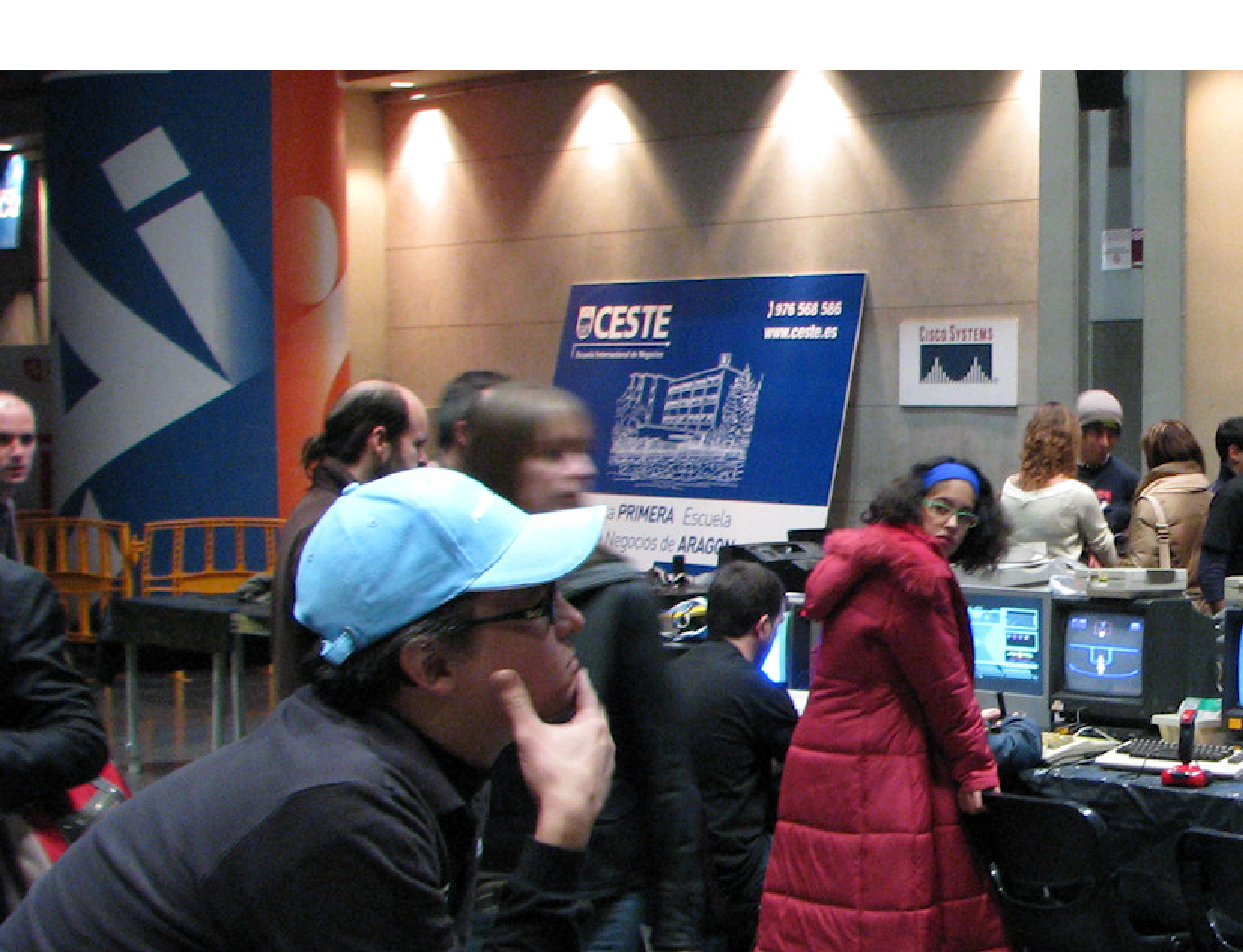
La traca final

RetroAcción cuenta ya con una larga tradición de organizar mesas redondas donde se debata sobre Retroinformática. Zaragoza no podía ser menos, y el sábado a las 20:00 se desarrolló "Obsoletos y orgullosos: las múltiples caras de la RetroInformática".

Como ponentes participaron:

- José Leandro Novellón, desarrollador de hardware para ordenadores de 8 bits, miembro del colectivo Speccy.org.
- Eduardo Mena (profesor titular del Departamento de Informática e Ingeniería de Sistemas, Universidad de Zaragoza).
- Ángel Ló (comunicador, miembro del colectivo Computer Emuzone y grafista dentro del sello amateur de videojuegos CEZ Games Studio).
- Iván Lalaguna (Presidente de la asociación DIR-Europa y coorganizador del museo de Informática del Centro Politécnico Superior, Universidad de Zaragoza).

La moderación corrió a cargo de Iñaki Grao, presidente de la Asociación RetroAcción. Durante 90 minutos se habló de las diferentes formas de disfrutar y entender esta afición: creación de nuevo hardware y software y preservación del antiguo, generación de nuevos contenidos y reuniones, planteamiento desde un punto de



vista académico, disfrute vegetativo... Como es habitual, la participación del público en forma de preguntas enriqueció el debate, que para variar, se hizo corto.

Echando el cierre

Si RetroMañía ha llegado a buen puerto, es gracias a la colaboración de los aficionados a la retroinformática de Zaragoza. Ellos, y la gente que vino de Bilbao y Madrid hicieron posible que con sólo quince días se pudiera dar contenido al evento. Esperamos que el camino que hemos tomado tenga continuidad en futuras ediciones y que sirva para dignificar una afición que hasta ahora, en su mayor parte, es algo desconocido.

En caso de que te hayas quedado con ganas de ver más, no dejes de visitar las webs de RetroMañía y RetroAcción, donde se irán publicando más contenidos en un futuro.

Links

- Retroeuskal:
  http://www.retroeuskal.org/
- Asociación RetroAcción:
  http://www.retroaccion.org/
- Red@ton:
  http://www.redaton.es/

Javier Vispe



programación ensamblador

lenguaje ensamblador del z80 (III)

Una vez hemos visto la mayoría de instrucciones aritméticas y lógicas, es el momento de utilizarlas como condicionales para realizar cambios en el flujo lineal de nuestro programa. En esta entrega aprenderemos a usar etiquetas y saltos mediante instrucciones condicionales (CP, JR + condición, JP + condición, etc.), lo que nos permitirá implementar en ensamblador las típicas instrucciones IF/THEN/ELSE y los GOTO de BASIC.

Las etiquetas en los programas ASM

Las etiquetas son unas directivas de los ensambladores (muy útiles) que nos permitirán hacer referencia a posiciones concretas de memoria, tanto dentro de nuestro programa como en otras áreas de memoria, por medio de nombres, en lugar de tener que utilizar valores numéricos.

Veamos un ejemplo de etiqueta en un programa ensamblador:

```
        ORG 50000

        NOP
        LD B, 10
bucle:
        LD A, 20
        NOP
        (...)
        JP bucle
        RET
```

Este es el código binario que genera el listado anterior al ser ensamblado:

```
00 06 0a 3e 14 00 (...) c3 53 c3 c9
```

Concretamente:

```
DIRECCION    OPCODE       INSTRUCCION
50000        00           NOP
50001        06 0a        LD B, 10
50003        3e 14        LD A, 20
50004        00           NOP
…            c3 53 c3     JP C353 (53000)
…+1    c9    RET
```

Si mostramos las direcciones de memoria en que se ensambla cada instrucción, veremos:

```
50000   NOP        ; (opcode = 1 byte)
50001   LD B, 10   ; (opcode = 2 bytes)
50003   LD A, 20   ; (opcode = 2 bytes)
50005   NOP        ; (opcode = 1 byte)
50005       (más código)
50006       (más código)
.....
50020   JP bucle
50023   RET
```

¿Dónde está en ese listado de instrucciones nuestra etiqueta "bucle"? Sencillo: no está. No es ninguna instrucción, sino, para el ensamblador, una referencia a la celdilla de memoria 50003, donde está la instrucción que sigue a la etiqueta.

En nuestro ejemplo anterior, mediante ORG 50000 le decimos al programa ensamblador que nuestro código, una vez ensamblado, pretendemos que quede situado a partir de la dirección 50000, con lo cual cuando calcule las direcciones de las etiquetas deberá hacerlo en relación a esta dirección de origen. Así, en nuestro ejemplo anterior la instrucción NOP, que se ensambla con el opcode 00h, será "pokeada" (por nuestro cargador BASIC) en la dirección 50000. La instrucción LD B, 10, cuyo opcode tiene 2 bytes, será "pokeada" en 50001 y 50002, y así con todas las instrucciones del programa.

Cuando el ensamblador se encuentra la etiqueta "bucle:" después del "LD B, 10", ¿cómo la ensambla? Supuestamente le corresponde la posición 50003, pero recordemos que esto no es una instrucción, sino una etiqueta: no tiene ningún significado para el microprocesador, sólo para el programa ensamblador. Por eso, cuando el ensamblador encuentra la etiqueta "bucle:", "mentalmente" asocia esta etiqueta (el texto "bucle") a la dirección 50003, que es la dirección donde hemos puesto la etiqueta.

Si la etiqueta fuera una instrucción, se ensamblaría en la dirección 50003, pero como no lo es, el programa ensamblador simplemente la agrega a una tabla interna de referencias, donde lo que anota es:

- La etiqueta "bucle" apunta a la dirección 50003

Lo que realmente ensamblará en la dirección 50003 (y 50004) es la instrucción siguiente: "LD A, 20".

Pero, entonces, ¿para qué nos sirve la etiqueta? Sencillo: para poder hacer referencia en cualquier momento a esa posición de memoria (del programa, en este caso), mediante una cadena fácil de recordar en lugar de mediante un número. Es más sencillo recordar "bucle" que recordar "50003", y si nuestro programa es largo y tenemos muchos saltos, funciones o variables, acabaremos manejando decenas y centenares de números para saltos, con lo que el programa sería inmanejable.

El siguiente programa es equivalente al anterior, pero sin usar etiquetas:

```
ORG 50000

NOP
LD B, 10
LD A, 20
NOP
(...)
JP 50003
RET
```

Las etiquetas son muy útiles. Veamos por qué: imaginemos que una vez acabado nuestro programa sin etiquetas (con direcciones numéricas), con muchos saltos (JP) a diferentes partes del mismo, tenemos que modificarlo porque hay un error. Al añadir o quitar instrucciones del programa, estamos variando las posiciones donde se ensambla todo el programa. Si por ejemplo, añadiéramos un NOP extra al principio del mismo, ya no habría que saltar a 50003 sino a 50004:

```
ORG 50000

NOP
NOP          ; Un NOP extra
LD B, 10
LD A, 20
NOP
(...)
JP 50004     ; La dirección de salto
               cambia
RET
```

Para que nuestro programa funcione, tendríamos que cambiar TODAS las direcciones numéricas de salto del programa, a mano (calculando todas). Las etiquetas evitan esto, ya que es el programa ensamblador quien, en tiempo de ensamblado, cuando está convirtiendo el programa a código objeto, cambia todas las referencias a la etiqueta por el valor numérico correcto (por la posición donde aparece la etiqueta).

Como veremos posteriormente, la instrucción JP realiza un salto de ejecución de código a una posición de memoria dada. Literalmente, un JP XX hace el registro PC = XX, de forma que alteramos el orden de ejecución del programa. Las etiquetas nos permiten establecer posiciones donde saltar en nuestro programa para utilizarlas luego fácilmente:

```
 ORG 50000

; Al salir de esta rutina, A=tecla pulsada
RutinaLeerTeclado:
   (instrucciones)    ; Aquí código
   RET

; Saltar (JP) a esta rutina con:
;  HL = Sprite a dibujar
;  DE = Direccion en pantalla donde dibujar
RutinaDibujarSprite:
   (...)
 bucle1:
     (instrucciones)
```

```
 bucle2:
     (instrucciones)
 pintar:
     (instrucciones)
     JP bucle1
   (...)
   RET

  (etc...)
```

Así, podremos especificar múltiples etiquetas para hacer referencia a todas las posiciones que necesitemos dentro de nuestro programa.

Por suerte, no sólo existen etiquetas para referenciar posiciones del programa. Además podemos insertar en cualquier posición de la memoria datos en formato numérico o de texto y hacer referencia a ellos, con comandos como DB (DEFB), DW (DEFW) o DS (DEFS). Por ejemplo:

```
 ; Principio del programa
 ORG 50000

   ; Primero vamos a copiar los datos a la videomemoria.
   LD HL, datos
   LD DE, 16384
   LD BC, 10
   LDIR

   ; Ahora vamos a sumar 1 a cada carácter:
   LD B, 27
bucle:
   LD HL, texto
   LD A, (HL)
   INC A
   LD (HL), A

   DJNZ bucle
   RET

datos DB 0, $FF, $FF, 0, $FF, 12, 0, 0, 0, 10, 255
texto DB "Esto es una cadena de texto"

   ; Fin del programa
   END
```

Como puede verse, con DB hemos "insertado" datos directamente dentro de nuestro programa. Estos datos se cargarán en memoria (pokeados) también como parte del programa, y podremos acceder a ellos posteriormente. Los datos, en nuestro programa, están situados en la memoria, justo después de las instrucciones ensambladas (tras el último RET). Podemos verlo si ensamblamos el programa:

```
$ pasmo --bin db.asm db.bin

00000000  21 66 c3 11 00 40 01 0a  00 ed b0 06 1b 21 71 c3  |!f...@.......!q.|
00000010  7e 3c 77 10 f8 c9 00 ff  ff 00 ff 0c 00 00 00 0a  |~
```

```
  10 CLEAR 49999
randomize usr 50000
  20 POKE 23610,255
  30 LOAD "" CODE

0 OK, 0:1
```

Resultado de RANDOMIZE USR 50000 en nuestro programa

Si os fijáis, podemos ver el RET (201, o $C9) justo antes del bloque de datos FF, FF, 0, FF. Concretamente, la etiqueta "datos" en el programa hará referencia (al pokear el programa a partir de 50000), a la posición de memoria 50022, que contendrá el 00 inicial de nuestros datos DB.

Cuando en el programa hacemos "LD HL, datos", el ensamblador transforma esa instrucción en realidad en "LD HL, 50022" (fijaos en el principio del programa: 21 66 C3, que corresponde a LD HL, C366, que es 50022). Gracias a esto podemos manipular los datos (que están en memoria) y leerlos y cambiarlos, utilizando un "nombre" como referencia a la celdilla de memoria de inicio de los mismos.

Lo mismo ocurre con el texto que se ha definido entre dobles comillas. A partir de la dirección definida por "texto" se colocan todos los bytes que forman la cadena "Esto es una cadena de texto". Cada byte en memoria es una letra de la cadena, en formato ASCII (La "E" es 45h, la "s" es 73h", etc.).

Con DB (o DEFB, que es equivalente por compatibilidad con otros ensambladores) podremos definir:

- Cadenas de texto (todos los mensajes de texto de nuestros programas/juegos).
- Datos numéricos con los que trabajar (bytes, words, caracteres...).
- Tablas precalculadas para optimizar. Por ejemplo, podemos tener un listado como el siguiente:

```
numeros_primos  DB  1, 3, 5, 7, 11, 13, (etc...)
```

- Variables en memoria para trabajar en nuestro programa:

```
   vidas  DB   3
   x      DB   0
   y      DB   0
   ancho  DB  16
   alto   DB  16
   (...)

   LD A, (vidas)
   (...)
muerte:
   DEC A
   LD (vidas), A
```

- Variables en memoria para trabajar en nuestro programa:

```
   vidas  DB   3
   x      DB   0
   y      DB   0
   ancho  DB  16
   alto   DB  16
   (...)

   LD A, (vidas)
```

```
   (...)
muerte:
   DEC A
   LD (vidas), A
```

• Datos gráficos de nuestros sprites (creados con utilidades como SevenuP o ZXPaintBrush, por ejemplo):

```
 Enemigo  DB 12, 13, 25, 123, 210 (etc...)
```

Ahora bien, es muy importante tener clara una consideración: los datos que introducimos con DB (o DW, o cualquier otra directiva de inclusión) no se ensamblan, pero se insertan dentro del código resultante tal cual. Y el Z80 no puede distinguir un 201 introducido con DB de un opcode 201 (RET), con lo cual tenemos que asegurarnos de que dicho código no se ejecute, como en el siguiente programa:

```
 ORG 50000

; Cuidado, al situar los datos aquí, cuando saltemos a 50000
; con RANDOMIZE USR 50000, ejecutaremos estos datos como si
; fueran opcodes.
datos DB 00, 201, 100, 12, 255, 11

  LD B, A
  (más instrucciones)
  RET
```

Lo correcto sería:

```
  ORG 50000

  ; Ahora el salto a 50000 ejecutará el LD B, A, no los
  ; datos que habíamos introducido antes.
  LD B, A
  (más instrucciones)
  RET

; Aquí nunca serán ejecutados, el RET está antes.
datos DB 00, 201, 100, 12, 255, 11
```

Los microprocesadores como el Z80 no saben distinguir entre datos e instrucciones, y es por eso que tenemos que tener cuidado de no ejecutar datos como si fueran códigos de instrucción del Z80. De hecho, si hacemos un RANDOMIZE USR XX (siendo XX cualquier valor de la memoria), lo más probable es que ejecutemos datos como si fueran instrucciones y el Spectrum se cuelgue, ya que los datos no son un programa, y el programa resultante de interpretar esos datos no tendría ningún sentido.

Lo que nos tiene que quedar claro de este apartado son dos conceptos: cuando el ensamblador encuentra la definición de una etiqueta, guarda en una tabla interna la dirección de ensamblado de la siguiente instrucción a dicha etiqueta. Después, cada vez que hay una aparición de esa etiqueta en el código, sustituye la etiqueta por dicha dirección de memoria. Además, podemos utilizar la etiqueta incluso aunque la definamos después (más adelante) del código, ya que el ensamblador hace varias pasadas en la compilación: no es necesario primero definir la etiqueta y después hacer referencia a ella, podemos hacerlo también a la inversa.

Es decir, es válido tanto:

```
etiqueta:
    (más código)
    JP etiqueta
```

Como:

```
 JP etiqueta
 (más código)
etiqueta:
```

Por otra parte, podemos utilizar etiquetas junto a la directiva DB (también podríamos usar DB sin etiqueta) para definir variables, bloques de datos, gráficos, cadenas, y en general cualquier tipo de dato en crudo que queramos insertar dentro de nuestro programa.

Saltos absolutos incondicionales: JP

Ya sabemos definir etiquetas en nuestros programas y referenciarlas. Ahora la pregunta es: ¿para qué sirven estas etiquetas? Ya lo hemos visto: aparte de referencias para usarlas como variables o datos, su principal uso será saltar a ellas con las instrucciones de salto.

Para empezar vamos a ver 2 instrucciones de salto incondicionales, es decir, cuando lleguemos a una de esas 2 instrucciones, se modificará el registro PC para cambiar la ejecución del programa. ¿Qué quiere decir esto? Que podemos realizar bucles, saltos a rutinas o funciones, etc. Empecemos con JP (de JumP):

```
      ; Ejemplo de un programa con
un bucle infinito
      ORG 50000

bucle:
      INC A
      LD (16384), A
      JP bucle

      RET     ; Esto nunca se
ejecutará
```

¿Qué hace el ejemplo anterior? Ensamblémoslo con "pasmo –tapbas bucle.asm bucle.tap" y ejecutémoslo en BASIC con un RUN y RANDOMIZE USR 50000 una vez cargado. Nada más entrar en 50000, se ejecuta un "INC A". Después se hace un "LD (16384), A", es decir, escribimos en la celdilla (16384) de la memoria el valor que contiene A. Esta celdilla se corresponde con los primeros 8 píxeles de la pantalla, con lo cual estaremos cambiando el contenido de la misma.

Tras esta escritura, encontramos un "JP bucle", que lo que hace es cambiar el valor de PC y hacerlo, de nuevo, PC=50000 (como hizo el RANDOMIZE USR 50000). El código se volverá a repetir, y de nuevo al llegar a JP volveremos a saltar a la dirección definida por la etiqueta "bucle". Es un bucle infinito, realizado gracias a este salto condicional (podemos reiniciar el Spectrum para retomar el control). Estaremos repitiendo una y otra vez la misma porción de código, que cambia el contenido de los 8 primeros píxeles de pantalla poniendo en ellos el valor de A (que varía desde 0 a 255 continuadamente).

Utilizaremos pues JP para cambiar el rumbo del programa y cambiar PC para ejecutar otras porciones de código (anteriores o posteriores a la posición actual) del mismo. JP realiza pues lo que se conoce como "SALTO INCONDICIONAL ABSOLUTO", es decir, saltar a una posición absoluta de memoria (una celdilla de 0 a 65535), mediante la asignación de dicho valor al registro PC.

Existen 3 maneras de usar JP:

a. JP NN:
 Saltar a la dirección NN. Literalmente: PC = NN

b. JP (HL)
 Saltar a la dirección contenida en el registro HL (ojo, no a la dirección apuntada por el registro HL, sino directamente a su valor). Literalmente: PC = HL

c. JP (registro_indice)
 Saltar a la dirección contenida en IX o IY. Literalmente: PC = IX o PC = IY

Ninguna de estas 4 instrucciones afecta a los flags:

	Flags					
Instrucción	S	Z	H	P	N	C
JP NN	-	-	-	-	-	-
JP (HL)	-	-	-	-	-	-
JP (IX)	-	-	-	-	-	-
JP (IY)	-	-	-	-	-	-

A la hora de ensamblar un salto como "JP 50000" (JP C350), dicha instrucción se ensamblará como:

```
C3 50 C3
```

que quiere decir:

```
JP 50 C3   ->   JP C350   ->   JP 50000
```

(Recordemos que nuestra CPU almacena primero en memoria los bytes bajos de los números de 16 bits). Como podéis ver, aparte del código de instrucción (C3) almacenamos un valor numérico, absoluto, de la posición a la que saltar. Es pues una instrucción de 3 bytes.

Literalmente, JP NN se traduce por PC=NN.

Saltos relativos incondicionales: JR

Además de JP, tenemos otra instrucción para realizar saltos incondicionales: JR. JR trabaja exactamente igual que JP: realiza un salto (cambiando el valor del registro PC), pero lo hace de forma diferente.

JR son las siglas de "Jump Relative", y es que esta instrucción en lugar de realizar un salto absoluto (a una posición de memoria 0-65535), lo hace de forma relativa, es decir, a una posición de memoria alrededor de la posición actual (una vez decodificada la instrucción JR). El argumento de JR no es pues un valor numérico de 16 bits (0-65535) sino un valor de 8 bits en complemento a dos que nos permite saltar desde la posición actual (referenciada en el ensamblador como "$") hasta 127 bytes hacia adelante y 128 bytes hacia atrás:

Ejemplos de instrucciones JR:

```
 JR $+25       ; Saltar adelante 25 bytes: PC = PC+25
 JR $-100      ; Saltar atrás 100 bytes:   PC = PC-100
```

Nosotros, gracias a las etiquetas, podemos olvidarnos de calcular posiciones y hacer referencia de una forma más sencilla a posiciones en nuestro programa:

Veamos el mismo ejemplo anterior de JP, con JR:

```
      ; Ejemplo de un programa con un bucle infinito
      ORG 50000

bucle:
      INC A
      LD (16384), A
      JR bucle

      RET ; Esto nunca se ejecutará
```

Como puede verse, el ejemplo es exactamente igual que en el caso anterior. No tenemos que utilizar el carácter $ (posición actual de ensamblado) porque al hacer uso de etiquetas es el ensamblador quien se encarga de traducir la etiqueta a un desplazamiento de 8 bits y ensamblarlo.

¿Qué diferencia tiene JP con JR? Pues bien: para empezar en lugar de ocupar 3 bytes (JP + la dirección de 16 bits), ocupa sólo 2 (JR + el desplaziento de 8 bits) con lo cual se decodifica y ejecuta más rápido.

Además, como la dirección del salto no es absoluta, sino relativa, y de 8 bits en complemento a dos, no podemos saltar a cualquier punto del programa, sino que sólo podremos saltar a código que esté cerca de la línea actual: como máximo 128 bytes por encima o 127 bytes por debajo de la posición actual en memoria.

¿Cuál es la utilidad o ventaja de esto? Pues que los saltos realizados en rutinas que usen JR y no JP son todos relativos a la posición actual, con lo cual la rutina es REUBICABLE. Es decir, si cambiamos nuestra rutina de 50000 a 60000 (por ejemplo), funcionará, porque los saltos son relativos a "$". En una rutina

programada con JP, si la pokeamos en 60000 en lugar de en 50000, cuando hagamos saltos (JP 50003, por ejemplo), saltaremos a lugares donde no está el código (ahora está en 60003) y el programa no hará lo que esperamos. En resumen: JR permite programar rutinas reubicables y JP no.

(Nota: se dice que una rutina es reubicable cuando estando programada a partir de una determinada dirección de memoria, podemos copiar la rutina a otra dirección y que sus saltos funcionen correctamente por no ser absolutos).

¿Recordáis en los cursos y rutinas de Microhobby cuando se decía "Esta rutina es reubicable"? Pues quería decir exactamente eso, que podías copiar la rutina en cualquier lugar de la memoria y llamarla, dado que el autor de la misma había utilizado sólo saltos relativos y no absolutos.

En nuestro caso, al usar un programa ensamblador en lugar de simplemente disponer de las rutinas en código máquina (ya ensambladas) que nos mostraba microhobby, no se nos plantearán esos problemas, dado que nosotros podemos usar etiquetas y copiar cualquier porción del código a dónde queramos de nuestro programa. Aquellas rutinas utilizaban direcciones de memoria numéricas o saltos relativos.

Nuestro ensamblador (Pasmo, o el que sea) nos permite utilizar etiquetas, que serán reemplazadas por sus direcciones de memoria durante el proceso de ensamblado. Nosotros podemos modificar las posibles de nuestras rutinas en el código, y dejar que el ensamblador las "reubique" por nosotros, ya que al ensamblará cambiará todas las referencias a las etiquetas que usamos.

Esta facilidad de trabajo contrasta con las dificultades que tenían los programadores de la época que no disponían de ensambladores profesionales. Imaginad la cantidad de usuarios que ensamblaban sus programas a mano, usando etiquetas, saltos absolutos (y como veremos, llamadas a subrutinas), que en lugar de sencillos nombres (JP A_mayor_que_B) utilizaban directamente direcciones en memoria. E imaginad el trabajo que suponía mantener un listado en papel de todas los saltos, subrutinas y variables, referenciados por direcciones de memoria y no por nombres, y tener que cambiar muchos de ellos cada vez que tenían que arreglar un fallo en una subrutina y cambiaban los destinos de los saltos por crecer el código que había entre ellos.

Dejando ese tema aparte, a tabla de afectación de flags de JR es la misma que para JP: nula.

	Flags					
Instrucción	S	Z	H	P	N	C
JR d	-	-	-	-	-	-

Donde "d" es un desplazamiento de 8 bits.

Literalmente, JR d se traduce por PC=PC+d.

Saltos condicionales con los flags

Hemos visto la forma de realizar saltos incondicionales. A continuación veremos cómo realizar los saltos (ya sean absolutos con JP o relativos con JR) de acuerdo a unas determinadas condiciones.

Las instrucciones condicionales disponibles trabajan con el estado de los flags del registro F, y son:

- JP NZ, direccion : Salta si el indicador de cero (Z) está a cero (resultado no cero).
- JP Z, direccion : Salta si el indicador de cero (Z) está a uno (resultado cero).
- JP NC, direccion : Salta si el indicador de carry (C) está a cero.
- JP C, direccion : Salta si el indicador de carry (C) está a uno.
- JP PO, direccion : Salta si el indicador de paridad/desbordamiento (P/O) está a cero.
- JP PE, direccion : Salta si el indicador de paridad/desbordamiento (P/O) está a uno.
- JP P, direccion : Salta si el indicador de signo S está a cero (resultado positivo).
- JP M, direccion : Salta si el indicador de signo S está a uno (resultado negativo).
- JR NZ, relativo : Salta si el indicador de cero (Z) está a cero (resultado no cero).
- JR Z, relativo : Salta si el indicador de cero (Z) está a uno (resultado cero).
- JR NC, relativo : Salta si el indicador de carry (C) está a cero.
- JR C, relativo : Salta si el indicador de carry (C) está a uno.

Donde "dirección" es un valor absoluto 0-65535, y

"relativo" es un desplazamiento de 8 bits con signo -128 a +127.

(Nota: en el listado de instrucciones, positivo o negativo se refiere a considerando el resultado de la operación anterior en complemento a dos).

Así, supongamos el siguiente programa:

```
    JP Z, bucle
    LD A, 10
bucle:
    NOP
```

(donde "bucle" es una etiqueta definida en algún lugar de nuestro programa, aunque también habríamos podido especificar directamente una dirección como por ejemplo 50004).

Cuando el procesador lee el "JP Z, bucle", lo que hace es lo siguiente:

• Si el flag Z está activado (a uno), saltamos a "bucle" (con lo cual no se ejecuta el "LD A, 10"), ejecutándose el código a partir del "NOP".
• Si no está activo (a cero) no se realiza ningún salto, con lo que se ejecutaría el "LD A, 10", y seguiría después con el "NOP".

En BASIC, "JP Z, bucle" sería algo como:

```
IF FLAG_ZERO = 1 THEN GOTO bucle
```

Y "JP NZ, bucle" sería:

```
IF FLAG_ZERO = 0 THEN GOTO bucle
```

Con estas instrucciones podemos realizar saltos condicionales en función del estado de los flags o indicadores del registro F: podemos saltar si el resultado de una operación es cero, si no es cero, si hubo acarreo, si no lo hubo...

Y el lector se preguntará: ¿y tiene utilidad para mí realizar saltos en función de los flags? Pues la respuesta es: bien usados, lo tiene para todo tipo de tareas:

```
  ; Repetir 100 veces la instruccion NOP
  LD A, 100
bucle:
  NOP

  DEC A          ; Decrementamos A.
                 ; Cuando A sea cero, Z se pondrá a 1
  JR NZ, bucle   ; Mientras Z=0, repetir el bucle
  LD A, 200      ; Aquí llegaremos cuando Z sea 1 (A valga 0)
  ; resto del programa
```

Es decir: cargamos en A el valor 100, y tras hacer nuestro "NOP", hacemos un "DEC A" que decrementa su valor (a 99). Como el resultado de "DEC A" es 99 y no cero, el flag de Z (de cero) se queda a 0, (recordemos que sólo se pone a uno cuando la última operación resultó ser cero).

Como el flag Z es cero (NON ZERO = no activado el flag zero) la instrucción "JR NZ, bucle" realiza un salto a la etiqueta "bucle". Allí se ejecuta el NOP y de nuevo el "DEC A", dejando ahora A en 98.

Tras repetirse 100 veces el proceso, llegará un momento en que A valga cero tras el "DEC A". En ese momento se activará el flag de ZERO con lo que la instrucción "JR NZ, bucle" no realizará el salto y continuará con el "LD A, 200".

Veamos otro ejemplo más gráfico: vamos a implementar en ASM una comparación de igualdad:

```
IF A=B THEN GOTO iguales ELSE GOTO distintos
```

En ensamblador:

```
    SUB B              ; A = A-B
    JR Z, iguales      ; Si Z=1 saltar a iguales
    JR NZ, distintos   ; Si Z=0 saltar a distintos

iguales:
    (código)
    JR seguir

distintos:
    (código)
    JR seguir

seguir:
```

(Nota: se podría haber usado JP en vez de JR)

Para comparar A con B los restamos (A=A-B). Si el resultado de la resta es cero, es porque A era igual a B. Si no es cero, es que eran distintos. Y utilizando el flag de Zero con JP Z y JP NZ podemos detectar esa diferencia.

Pronto veremos más a fondo otras instrucciones de comparación, pero este ejemplo debe bastar para demostrar la importancia de los flags y de su uso en instrucciones de salto condicionales. Bien utilizadas podemos alterar el flujo del programa a voluntad. Es cierto que no es tan inmediato ni cómodo como los >, <, = y <> de BASIC, pero el resultado es el mismo, y es fácil acostumbrarse a este tipo de comparaciones mediante el estado de los flags.

Para finalizar, un detalle sobre DEC+JR: La combinación DEC B / JR NZ se puede sustituir (es más eficiente, y más sencillo) por el comando DJNZ, que literalmente significa "Decrementa B y si no es cero, salta a ".

DJNZ dirección

Esta instrucción se usa habitualmente en bucles (usando B como iterador del mismo) y, al igual que JP y JR, no afecta al estado de los flags:

	Flags					
Instrucción	S	Z	H	P	N	C
JP COND, NN	-	-	-	-	-	-
JR d	-	-	-	-	-	-
DJNZ d	-	-	-	-	-	-

Instrucción de comparación CP

Para realizar comparaciones (especialmente de igualdad, mayor que y menor que) utilizaremos la instrucción CP. Su formato es:

```
CP origen
```

Donde "origen" puede ser A, F, B, C, D, E, H, L, un valor numérico de 8 bits directo, (HL), (IX+d) o (IY+d).

Al realizar una instrucción "CP origen", el microprocesador ejecuta la operación "A-origen", pero no almacena el resultado en ningún sitio. Lo que sí que hace es alterar el estado de los flags de acuerdo al resultado de la operación.

Recordemos el ejemplo de comparación anterior donde realizábamos una resta, perdiendo por tanto el valor de A:

```
    SUB B              ; A = A-B
    JR Z, iguales      ; Si Z=1 saltar a iguales
    JR NZ, distintos   ; Si Z=0 saltar a distintos
```

Gracias a CP, podemos hacer la misma operación pero sin perder el valor de A (por la resta):

```
    CP B                ; Flags = estado(A-B)
    JR Z, iguales       ; Si Z=1 saltar a iguales
    JR NZ, distintos    ; Si Z=0 saltar a distintos
```

¿Qué nos permite esto? Aprovechando todos los flags del registro F (flag de acarreo, flag de zero, etc), realizar comparaciones como las siguientes:

```
   ; Comparación entre A Y B (=, > y <)
   LD B, 5
   LD A, 3

   CP B                     ; Flags = estado(A-B)
   JP Z, A_Igual_que_B      ; IF(a-b)=0 THEN a=b
   JP NC, A_Mayor_que_B     ; IF(a-b)>0 THEN a>b
   JP C, A_Menor_que_B      ; IF(a-b)<0 THEN a
```

Vamos a ilustrar la anterior porción de código con un ejemplo que nos permitirá, además, descubrir una forma muy singular de hacer debugging en vuestras pruebas aprendiendo ensamblador. Vamos a sacar información por pantalla de forma que podamos ver en qué parte del programa estamos. Este mismo "sistema" podéis emplearlo (hasta que veamos cómo sacar texto o gráficos concretos por pantalla) para "depurar" vuestros programas y hacer pruebas.

Consiste en escribir valor en la memoria, justo en la zona de la pantalla, para así distinguir las partes de nuestro programa por las que pasamos. Así, escribiremos 255 (8 pixeles activos) en una línea de la parte superior de la pantalla izquierda (16960), en el centro de la misma (19056), o en la parte inferior derecha (21470):

```
  ; Principio del programa
  ORG 50000

   ; Comparacion entre A Y B (=, > y <)
   LD B, 7
   LD A, 5

   CP B                     ; Flags = estado(A-B)
   JP Z, A_Igual_que_B      ; IF(a-b)=0 THEN a=b
   JP NC, A_Mayor_que_B     ; IF(a-b)>0 THEN a>b
   JP C, A_Menor_que_B      ; IF(a-b)<0 THEN a
```

Lo ensamblamos con: pasmo –tapbas compara.asm compara.tap, y lo cargamos en el Spectrum o emulador. La sentencia END 50000 nos ahorra el teclear "RANDOMIZE USR 50000" ya que pasmo lo introducirá en el cargador BASIC por nosotros. Jugando con los valores de A y B del listado deberemos ver cómo cambia el lugar al que saltamos (representado por el lugar de la pantalla en que vemos dibujada nuestra pequeña línea de 8 píxeles).

Bytes: compara.ta

Resultado de RANDOMIZE USR 50000 en nuestro programa

Finalmente, destacar que nada nos impide el hacer comparaciones multiples o anidadas:

```
  LD B, 5
  LD A, 3
  LD C, 6

  CP B                   ; IF A==B
  JR Z, A_Igual_a_B      ; THEN goto A_Igual_a_B
  CP C                   ; IF A==C
  JR Z, A_Igual_a_C      ; THEN goto A_Igual_a_C
  JP Fin                 ; si no, salimos
A_Igual_a_B:
  (...)
  JR Fin

A_Igual_a_C:
  (...)

Fin:
  (resto del programa)
```

La instrucción CP afecta a todos los flags:

	Flags					
Instrucción	S	Z	H	P	N	C
CP s	*	*	*	V	1	*

El flag "N" se pone a uno porque, aunque se ignore el resultado, la operación efectuada es una resta.

Consideraciones de las condiciones

A la hora de utilizar instrucciones condicionales hay que tener en cuenta que no todas las instrucciones afectan a los flags. Por ejemplo, la instrucción "DEC BC" no pondrá el flag Z a uno cuando BC sea cero. Si intentamos montar un bucle mediante DEC BC + JR NZ, nunca saldremos del mismo, ya que DEC BC no afecta al flag de zero.

```
    LD BC, 1000         ; BC = 1000
bucle:
    (...)

    DEC BC              ; BC = BC-1 (pero NO ALTERA el Carry Flag)
    JR NZ, bucle        ; Nunca se pondrá a uno el ZF, siempre salta
```

Para evitar estas situaciones necesitamos conocer la afectación de los flags ante cada instrucción, que podéis consultar en todas las tablas que os hemos proporcionado.

Podemos realizar algo similar al ejemplo anterior aprovechándonos (de nuevo) de los flags y de los resultados de las operaciones lógicas (y sus efectos sobre el registro F):

```
    LD BC, 1000         ; BC = 1000

bucle:
    (...)
    DEC BC              ; Decrementamos BC. No afecta a F.
    LD A, B             ; A = B
    OR C                ; A = A OR C
                        ; Esto sí que afecta a los flags.
                        ; Si B==C y ambos son cero, el resultado
                        ; del OR será cero y el ZF se pondrá a 1.
    JR NZ, bucle        ; ahora sí que funcionará el salto si BC=0
```

Más detalles sobre los saltos condicionales: esta vez respecto al signo. Las condiciones P y M (JP P, JP M) nos permitirán realizar saltos según el estado del bit de signo. Resultará especialmente útil después de operaciones aritméticas.

Los saltos por Paridad/Overflow (JP PO, JP PE) permitirán realizar saltos en función de la paridad cuando la última operación realizada modifique ese bit de F según la paridad del resultado. La misma condición nos servirá para desbordamientos si la última operación que afecta a flags realizada modifica este bit con respecto a dicha condición. ¿Qué quiere decir esto? Que si, por ejemplo, realizamos una suma o resta, JP PO y JP PE responderán en función de si ha habido un desbordamiento o no y no en función de la paridad, porque las sumas y restas actualizan dicho flag según los desbordamientos, no según la paridad.

La importancia de la probabilidad de salto

Ante una instrucción condicional, el microprocesador tendrá 2 opciones, según los valores que comparemos y el tipo de comparación que hagamos (si es cero, si no es cero, si es mayor o menor, etc.). Al final, sólo habrá 2 caminos posibles: saltar a una dirección de destino, o no saltar y continuar en la dirección de memoria siguiente al salto condicional.

Aunque pueda parecer una pérdida de tiempo, en rutinas críticas es muy interesante el pararse a pensar cuál puede ser el caso con más probabilidades de ejecución, ya que el tiempo empleado en la opción "CONDICION CIERTA, POR LO QUE SE PRODUCE EL SALTO" es mayor que el empleado en "CONDICION FALSA, NO SALTO Y SIGO".

Por ejemplo, ante un "JP Z, direccion", el microprocesador tardará 10 ciclos de reloj en ejecutar un salto si la condición se cumple, y sólo 1 si no se cumple (ya que entonces no tiene que realizar salto alguno).

Supongamos que tenemos una rutina que recorre una zona de memoria y lee los diferentes valores de 0 a 255. Si el valor es mayor de 240 (es decir, en el intervalor 241-255), lo pone a cero, mientras que si es menor de 240 (en el intervalo 0-240), lo cambia por un 1.

Lo normal es que, ante datos aleatorios, haya más probabilidad de encontrar datos del segundo caso (0-240) que del primero (241-255), simplemente por el hecho de que hay 240 probabilides de 255, mientras que del primero hay 15 probabilidades de 255. En tal caso, dicha rutina debería organizarse de forma que la comparación realice el salto cuando encuentre un dato mayor de 255, dado que ese supuesto se dará menos veces. Si lo hicieramos a la inversa, se saltaría más veces y la rutina tardaría más en realizar el mismo trabajo.

Instrucciones de comparacion repetitivas

Para acabar con las instrucciones de comparación vamos a ver las instrucciones de comparación repetitivas. Son parecidas a CP, pero trabajan (igual que LDI, LDIR, LDD y LDDR) con HL y BC para realizar las comparaciones con la memoria: son CPI, CPD, CPIR y CPDR.

Comencemos con CPI (ComPare and Increment):

CPI:

- Al registro A se le resta el byte contenido en la posición de memoria apuntada por HL.
- El resultado de la resta no se almacena en ningún sitio.
- Los flags resultan afectados por la comparación:
  - Si A==(HL), se pone a 1 el flag de Zero (si no es igual se pone a 0).
  - Si BC==0000, se pone a 0 el flag Parity/Overflow (a 1 en caso contrario).
- Se incrementa HL.
- Se decrementa BC.

Técnicamente (con un pequeño matiz que veremos ahora), CPI equivale a:

```
CPI =     CP [HL]
          INC HL
          DEC BC
```

Su instrucción "hermana" CPD (ComPare and Decrement) funciona de idéntica forma, pero decrementando HL:

```
CPD =     CP [HL]
          DEC HL
          DEC BC
```

Y el pequeño matiz: así como CP [HL] afecta al indicador C de Carry, CPI y CPD, aunque realizan esa operación intermedia, no lo afectan. Las instrucciones CPIR y CPDR son equivalentes a CPI y CPD, pero ejecutándose múltiples veces: hasta que BC sea cero o bien se encuentre en la posición de memoria apuntada por HL un valor numérico igual al que contiene el registro A. Literalmente, es una instrucción de búsqueda: buscamos hacia adelante (CPIR) o hacia atrás (CPDR), desde una posición de memoria inicial (HL), un valor (A), entre dicha posición inicial (HL) y una posición final (HL+BC o HL-BC para CPIR y CPDR).

CPIR:

• Al registro A se le resta el byte contenido en la posición de memoria apuntada por HL.
• El resultado de la resta no se almacena en ningún sitio.
• Los flags resultan afectados por la comparación:
  * Si A==(HL), se pone a 1 el flag de Zero (si no es igual se pone a 0).
  * Si BC==0000, se pone a 0 el flag Parity/Overflow (a 1 en caso contrario).
• Se incrementa HL.
• Se decrementa BC.
• Si BC===0 o A=(HL), se finaliza la instrucción. Si no, repetimos el proceso.

CPDR es, como podéis imaginar, el equivalente a CPIR pero decrementando HL, para buscar hacia atrás en la memoria.

Como ya hemos comentado, muchos flags se ven afectados:

	Flags					
Instrucción	S	Z	H	P	N	C
CPI	*	*	*	*	1	
CPD	*	*	*	*	1	
CPIR	*	*	*	*	1	
CPDR	*	*	*	*	1	

Un ejemplo de uso de un CP repetitivo es realizar búsquedas de un determinado valor en memoria. Supongamos que deseamos buscar la primera aparición del valor "123" en la memoria a partir de la dirección 20000, y hasta la dirección 30000, es decir, encontrar la dirección de la primera celdilla de memoria entre 20000 y 30000 que contenga el valor 123.

Podemos hacerlo mediante el siguiente ejemplo con CPIR:

```
    LD HL, 20000       ; Origen de la busqueda
    LD BC, 10000       ; Número de bytes a buscar (20000-30000)
    LD A, 123          ; Valor a buscar
    CPIR
```

Este código realizará lo siguiente:

```
  HL = 20000
  BC = 10000
  A  = 123

CPIR =
Repetir:
  Leer el contenido de (HL)
  Si A==(HL) -> Fin_de_CPIR
  Si BC==0   -> Fin_de_CPIR
  HL = HL+1
  BC = BC-1
Fin_de_CPIR:
```

Con esto, si la celdilla 15000 contiene el valor "123", la instrucción CPIR del ejemplo anterior acabará su ejecución, dejando en HL el valor 15001 (tendremos que decrementar HL para obtener la posición exacta). Dejará además el flag "P/O" (paridad/desbordamiento) y el flag Z a uno. En BC tendremos

restado el número de iteraciones del "bucle" realizadas.

Si no se encuentra ninguna aparición de "123", BC llegará a valer cero, porque el "bucle CPI" se ejecutará 10000 veces. El flag P/O estará a cero, al igual que Z, indicando que se finalizó el CPIR y no se encontró nada.

Nótese que si en vez de utilizar CPIR hubiéramos utilizado CPDR, podríamos haber buscado hacia atrás, desde 20000 a 10000, decrementando HL. Incluso haciendo HL=0 y usando CPDR, podemos encontrar la última aparición del valor de A en la memoria (ya que 0000 - 1 = FFFFh, es decir: 0-1=65535 en nuestros 16 bits).

Un ejemplo con CPIR

Veamos un ejemplo práctico con CPIR. El código que veremos a continuación realiza una búsqueda de un determinado carácter ASCII en una cadena de texto:

```
 ; Principio del programa
 ORG 50000

   LD HL, texto      ; Inicio de la busqueda
   LD A, 'X'         ; Carácter (byte) a buscar
   LD BC, 100        ; Número de bytes donde buscar
   CPIR              ; Realizamos la búsqueda

   JP NZ, No_Hay     ; Si no encontramos el caracter buscado
                     ; el flag de Z estará a cero.

                     ; Si seguimos por aquí es que se encontró
   DEC HL            ; Decrementamos HL para apuntar al byte
                     ; encontrado en memoria.

   LD BC, texto
   SCF
   CCF               ; Ponemos el carry flag a 0 (SCF+CCF)
   SBC HL, BC        ; HL = HL - BC
                     ;    = (posicion encontrada) - (inicio cadena)
                     ;    = posición de 'X' dentro de la cadena.

   LD B, H
   LD C, L           ; BC = HL

   RET               ; Volvemos a basic con el resultado en BC

No_Hay:
   LD BC, $FFFF
   RET

texto DB "Esto es una X cadena de texto."

  ; Fin del programa
  END
```

Lo compilamos con "pasmo –tapbas buscatxt.asm buscatxt.tap", lo cargamos en el emulador y tras un RUN ejecutamos nuestra rutina como "PRINT AT 10,10 ; USR 50000". En pantalla aparecerá el valor 12.

¿Qué significa este "12"? Es la posición del carácter 'X' dentro de la cadena de texto. La hemos obtenido de la siguiente forma:

- Hacemos HL = posición de memoria donde empieza la cadena.

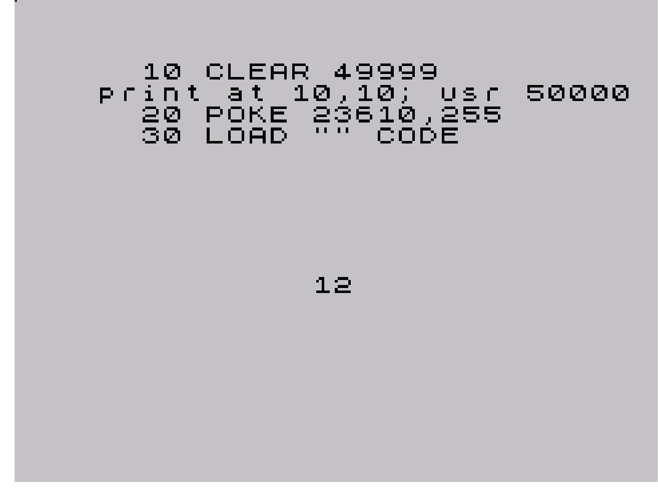
• Hacemos A = 'X'.
Ejecutamos un CPIR
En HL obtendremos la posición absoluta + 1 donde se encuentra el carácter 'X' encontrado (o FFFFh si no se encuentra). Exactamente 50041.
• Decrementamos HL para que apunte a la 'X' (50040).
• Realizamos la resta de Posicion('X') - PrincipioCadena para obtener la posición del carácter dentro de la cadena. De esta forma, si la 'E' de la cadena está en 50028, y la X encontrada en 50040, eso quiere decir que la 'X' está dentro del array en la posición 50040-50028 = 12.
• Volvemos al BASIC con el resultado en BC. El PRINT USR 50000 imprimirá dicho valor de retorno.

Nótese que el bloque desde "SCF" hasta "LD C, L" tiene como objetivo ser el equivalente a "HL = HL - BC", y se tiene que hacer de esta forma porque no existe "SUB HL, BC" ni "LD BC, HL":

```
SUB HL, BC =  SCF
              CCF             ; Ponemos el carry flag a 0 (SCF+CCF)
              SBC HL, BC      ; HL = HL - BC

LD BC, HL  =  LD B, H
              LD C, L         ; BC = HL
```

(Podemos dar las gracias por estas extrañas operaciones a la no ortogonalidad del juego de instrucciones del Z80).



Salida del programa buscatxt.asm

En resumen

En esta entrega hemos aprendido a utilizar todas las funciones condicionales y de salto de que nos provee el Z80.

En el próximo capítulo trataremos la PILA (Stack) del Spectrum, gracias a la cual podremos implementar en ensamblador el equivalente a GOSUB/RETURN de BASIC, es decir, subrutinas.

Ficheros

• La directiva DB utilizada en un ejemplo.
  http://magazinezx.speccy.org/revistas/16/src/05 _db.asm
• Fichero tap del ejemplo db.asm.
  http://magazinezx.speccy.org/revistas/16/src/05 _db.tap
• Ejemplo de bucle infinito con JP.
  http://magazinezx.speccy.org/revistas/16/src/05 _bucle.asm
• Fichero tap del ejemplo bucle.asm.
  http://magazinezx.speccy.org/revistas/16/src/05 _bucle.tap
• Búsqueda de cadenas de texto con CPIR.
  http://magazinezx.speccy.org/revistas/16/src/05 _buscatxt.asm
• Fichero tap del ejemplo buscatxt.asm.
  http://magazinezx.speccy.org/revistas/16/src/05 _buscatxt.tap
• Comparación <, >, =.
  http://magazinezx.speccy.org/revistas/16/src/05 _compara.asm
• Tap del ejemplo anterior.
  http://magazinezx.speccy.org/revistas/16/src/05 _compara.tap

Links

- Set de caracteres.
  http://www.worldofspectrum.org/ZXSpectrum128+3Manual/chapter8pt28.html
- Web del Z80.
  http://www.z80.info/
- Z80 Reference de WOS.
  http://www.worldofspectrum.org/faq/reference/z80reference.htm
- Z80 Reference de TI86.
  http://ti86.acz.org/z80 _ref.htm
- FAQ de Icarus Productions.
  http://icarus.ticalc.org/articles/z80 _faq.html
- Microfichas de CM de MicroHobby.
  http://www.speccy.org/trastero/cosas/Fichas/fichas.htm
- Tablas de ensamblado y t-estados (pulsar en z80.txt, z80 _reference.txt, z80time.txt).
  http://www.ticalc.org/pub/text/z80/
- Curso de ASM de z80.info.
  http://www.z80.info/lesson1.htm
- PASMO.
  http://www.arrakis.es/~ninsesabe/pasmo/

Santiago Romero



opinión

El mercado retroinformático o "están locos estos romanos"

Ya habíamos comentado anteriormente los tres principios fundamentales que rigen el mercado retroinformático y que definen todo lo que acontece en el mismo (podéis verlo en su enlace al final). Ahora, en este artículo queremos centrarnos en los perfiles de las personas que interactúan en dicho mercado.

Hoy en día, Internet ha facilitado la conexión entre las personas interesadas en un mismo tema, confluyendo esos intereses a través de páginas y foros. En este sentido, las páginas de compraventa, con Ebay a la cabeza, y el mercado retroinformático se han venido complementando desde hace ya unos pocos años. Sin embargo, si asistimos como espectadores a una función, vemos una serie de acontecimientos que nos llevan a exclamar, cual Astérix en Las Galias: "están locos estos romanos": Ventas que salen a precios desorbitados y que no se venden, ventas que acaban al doble de precio que otra semejante del día siguiente, ventas a bajo precio que tampoco se venden, pujadores frenéticos como si les fuera la vida, ventas que tienen una barbaridad de pujas de los mismos compradores, etc.

Pero lo que sucede, respetando fielmente los principios fundamentales que antes citábamos, no es más que una serie de hechos que vienen determinados por el perfil de ciertos agentes de ese mercado, cosa que vamos a detallar a continuación. Estos perfiles no son exhaustivos, sólo se consideran aquellos que no son los normales de comprador y vendedor en un mercado, y que son los que realmente determinan estos "raros hechos".

Por el lado de los vendedores tenemos:
- El vendedor "profesional". Basa su negocio en sus ventas, suele fijar un precio elevado que no baja si

no lo vende. Cuando se cansa de pagar comisiones en Ebay suele pasarlo a otras webs con menores comisiones. Se ofende si le ofreces un precio mas bajo y razonable.

- El vendedor "amoroso". Le tiene un cariño especial al objeto que vende, del cual sólo se podrá deshacer a un precio (generalmente muy alto) que le consuele. Como dice el refrán: "las penas con pan son menos".

- El vendedor "tacaño". Suele ingeniárselas para pagar las menores comisiones a Ebay, para ello fija un precio de venta muy bajo. Eso si, después las compras vienen acompañadas de elevados gastos de envío para compensar, y ni siquiera te lo envían certificado. Hay una versión mas moderna que después se autopuja para subir el precio mínimo. Si no aparece un comprador puede ganar su propia venta.

- El vendedor "tacaño y sinvergüenza". Versión del anterior que se busca excusas para no enviarte el artículo ganado si acabó a un precio que considere bajo. Tuve un caso que no encontraba en su casa donde había puesto el QL con su caja original incluida (50x35x15 cm).

- El vendedor "listillo". Él mismo o a través de un colega se dedica a calentar el precio de las pujas. Si ganaran la venta se la ofrecen al segundo comprador, ya que el ganador "no da señales de vida"; pero, claro está, al precio máximo que había pujado y perdido.

- El vendedor "revendedor". Suele fijar un precio alto, condicionado por el precio que ya ha pagado por el artículo que revende o bien por el diferencial de beneficio que quiere obtener. En este último caso suele ser un diferencial muy elevado, del orden de tres dígitos de incremento. Cuando mete la pata y compra a precio alto se lo suele comer con patatas.

Por el lado de los compradores tenemos:
- El comprador "compulsivo". Generalmente se calienta en webs de coleccionistas y foros "ad hoc", después entra como un elefante en una cacharrería, arrasando todo lo que se vende. Suele entrar en fase depre al cabo de 1 o 2 años, abandonando las compraventas e incluso llegando a vender su colección.

- El comprador "adinerado". No tiene problema para tirar de cartera y pagar lo que haga falta, vamos, ¿quién lo duda?. Peligroso cuando se junta con otro igual que él, pujando el uno contra el otro dejan enormes beneficios a los vendedores.

- El comprador "vendedor". Cual lobo envuelto en piel de cordero acude a sus ventas, se le reconoce por su bajo número de votos y porque puja eurito a eurito para no reventar la puja de otro comprador, haciéndole subir hasta el máximo que haya pujado. Si miras su perfil verás que siempre puja en las ventas de ese vendedor. A veces ganan su propia venta por subir tanto, entonces se la ofrecen al segundo pujador porque el ganador "tampoco da señales de vida". Parece que en ésto desaparece mucha gente.

- El comprador "revendedor". Es el mismo perfil del vendedor "revendedor", pero en su faceta compradora, busca ventas a bajo precio por lo que en esta fase no sube mucho el precio de mercado. No pujará nunca por encima de un precio que le permita maximizar su posterior venta, aunque algunas veces se equivoca y compra a un precio que no le permite revender.

- El comprador "despistado". Acude como un pingüino en el desierto, se le reconoce porque paga dos o tres veces más que el precio final de un artículo similar que haya acabado o acabe en los días siguientes.

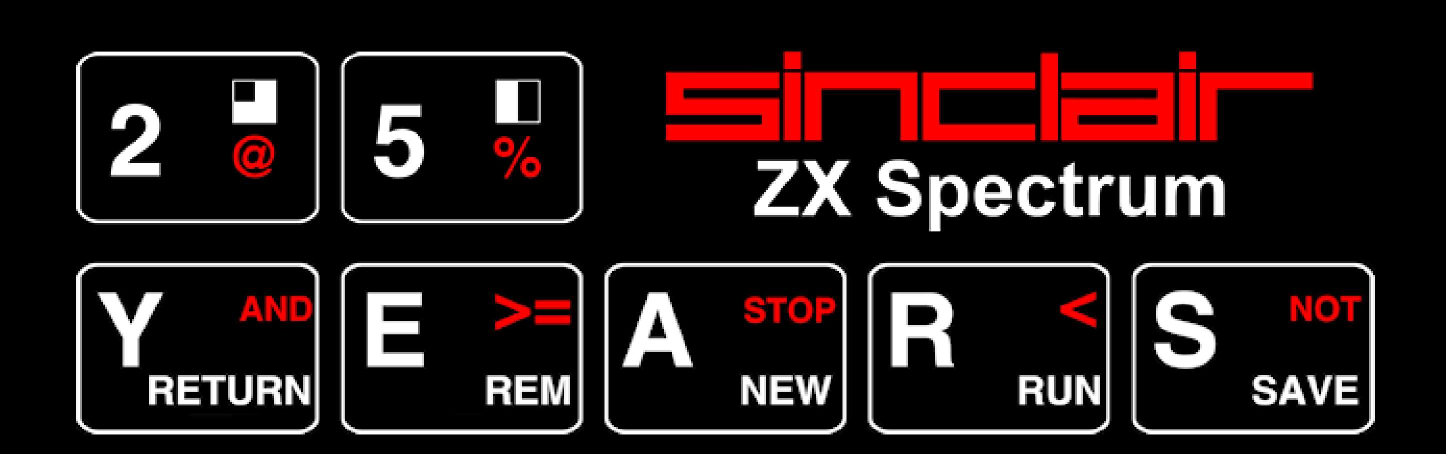
- El comprador "jodedor". Por diversos motivos se dedica a pujar y subir precios, nunca compra si gana, suele tener bajo perfil de votos e incluso votos negativos, también suele ser vendedor con otro usuario. Generalmente jode por joder o para hacer competitivas sus propias ventas.

En fin, la interrelación de todos estos elementos viene a explicar muchas compra-ventas que acontecen en Ebay y que pueden ser consideradas "extrañas". Por otro lado, tampoco son todos los que están ni están todos los que son, pero si los conocemos y los detectamos podremos tener ventas y compras "normales", con lo cual evitaremos hacer "el canelo" en muchas ocasiones. ¡Suerte!.

Links

- Los tres principios del precio de venta de un Spectrum: http://trastero.speccy.org/Principios.htm

José Manuel Claros


2
@
5
%
sinclair
ZX Spectrum
Y
AND
RETURN
E
>=
REM
A
STOP
NEW
R
<
RUN
S
NOT
SAVE